File: Account.java

```java
/**
 * Represents an account, located in the social media package
 * An account is made up of a user ID (uid), a handle a description field and a list of all posts it has made.
 *
 * @author Jeroen Mijer, Alexander Robertson
 * @version 1.0
 * @since 1.0
 */
package socialmedia;

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


public class Account implements Serializable {
    private Integer uid;
    private String handle;
    private String descField;
    private ArrayList<Post> posts;

    /**
     * Creates an account
     * Sets the arraylist of posts to empty, since an account can only make posts after it has been created
     *
     * @param handle code of the account
     * @param descField description field of the account
     * @param uid user ID of the account
     */
    public Account(String handle, String descField, Integer uid) {
        this.uid = uid;
        this.handle = handle;
        this.descField = descField;
        this.posts = new ArrayList<Post>();
    }


    /**
     * Gets account description field
     *
     * @return description field of the account
     */
    public String getDescField() {
        return descField;
    }
```

```java


/** Gets user id
 *
 * @return User ID of the account
 */
public Integer getUID() {
   return uid;
}


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


/** Gets arraylist of posts account has made
 *
 * @return Arraylist of posts account has made
 */
public ArrayList<Post> getPosts() {
   return posts;
}


/** Sets description field of account
 *
 * @param descField new description field of the account
 */
public void setDescField(String descField) {
   this.descField = descField;
}


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


```

```
    /** Adds post to arraylist of posts contained within account
     *
     * @param post Post being appended to the arraylist
     */
    public void addPost(Post post) {
        this.posts.add(post);
    }


    /** Removes original post from arraylist of posts contained within account
     *
     * @param post (OriginalPost) Post being removed from Arraylist
     */
    public void removePost(OriginalPost post) {
        this.posts.remove(post);
    }

    /** Removes comment from arraylist of posts contained within account
     *
     * @param post (Comment) Post being removed from Arraylist
     */
    public void removePost(Comment post) {
        this.posts.remove(post);
    }

    /** Removes endorsement from arraylist of posts contained within account
     *
     * @param post (Endorsement)Post being removed from Arraylist
     */
    public void removePost(Endorsement post) {
        this.posts.remove(post);
    }
}
```

File: Comment.java

```java
/**
 * Represents a comment, which extends the class Post, which is located in the social media package
 * An account is made up of a message, author, post ID (pid), arraylist of comments, arraylist of endorsements, a postpointer(points to parent post)
 *
 * @author Jeroen Mijer, Alexander Robertson
 * @version 1.0
 * @since 1.0
 */
package socialmedia;
import java.util.ArrayList;

public class Comment extends Post {

    Integer postPointer; // holds the postID that the comment is replying to
    ArrayList <Endorsement> endorsements;
    ArrayList <Comment> comments;

    /**
     * Creates a comment
     * Sets the arraylist of comments and endorsements to empty, since a comment can only be commented on or endorsed after creation
     * @param message message of the comment
     * @param author author of the comment
     * @param postPointer post ID of the parent post
     * @param pid post ID of the comment
     */
    public Comment(String message, Account author, Integer postPointer, Integer pid) {
        super(message, author, pid);
        this.endorsements = new ArrayList<Endorsement>();
        this.comments = new ArrayList<Comment>();
        this.postPointer = postPointer;
    }

    /** Gets post pointer
     *
     * @return post pointer of the comment
     */
    public Integer getPostPointer() {
        return postPointer;
    }


    /** Gets arraylist of endorsements
     *
     * @return arraylist of endorsements that endorsed a comment
     */
    public ArrayList<Endorsement> getEndorsements() {
        return endorsements;
    }


    /** Gets arraylist of comments
     *
     * @return arraylist of comments that commented on a comment
     */
    public ArrayList<Comment> getComments() {
        return comments;
    }


    /** Adds an endorsement to the arraylist of endorsements to a comment
     *
     * @param endorsement endorsement to be added to the arraylist of endorsements
     */
    public void addEndorsement(Endorsement endorsement) {
        this.endorsements.add(endorsement);
    }


    /** Adds a comment to the arraylist of comments in a comment
     *
     * @param comment comment to be added to the arraylist of endorsements
     */
    public void addComment(Comment comment) {
        this.comments.add(comment);
    }


    /** Sets post pointer of the comment
     * Used only when the parent comment is deleted and the post pointer now refers to the generic deleted message
     *
     * @param postPointer new postpointer that the comment will now contain
     */
    public void setPostPointer(Integer postPointer) {
        this.postPointer = postPointer;
    }
}
```

File: Endorsement.java

```java
/**
 * Represents an endorsement, which extends the class Post, which is located in the social media package
 * An endorsement is made up of a message, author, post ID (pid) and a postpointer(points to parent post)
 *
 * @author Jeroen Mijer, Alexander Robertson
 * @version 1.0
 * @since 1.0
 */
package socialmedia;
public class Endorsement extends Post {

    Integer postPointer; // holds the postID that the endorsement is endorsing

    /**
     * Creates an endorsement
     *
     * @param message message of the endorsement
     * @param author author of the endorsement
     * @param postPointer post ID of the parent post
     * @param pid post ID of the endorsement
     */
    public Endorsement(String message, Account author, Integer postPointer, Integer pid) {
        super(message, author, pid);
        this.postPointer = postPointer;

    }


    /** Gets the post pointer of an endorsement
     *
     * @return post pointer of an endorsement
     */
    public Integer getPostPointer() {
        return postPointer;
    }

}
```

File: OriginalPost.java

```java
/**
 * Represents an original post, which extends the class Post, which is located in the social media package
 * An original post is made up of a message, author, post ID (pid), arraylist of comments and an arraylist of endorsements
 *
 * @author Jeroen Mijer, Alexander Robertson
 * @version 1.0
 * @since 1.0
 */

package socialmedia;
import java.util.ArrayList;

public class OriginalPost extends Post {
    ArrayList <Endorsement> endorsements;
    ArrayList <Comment> comments;

    /**
     * Creates an original post
     * Sets the arraylist of comments and endorsements to empty, since an original post can only be commented on or endorsed after creation
     * @param message message of the original post
     * @param author author of the original post
     * @param pid post id of the original post
     */
    public OriginalPost(String message, Account author, Integer pid) {
        super(message, author, pid);
        this.endorsements = new ArrayList<Endorsement>();
        this.comments = new ArrayList<Comment>();

    }


    /** Gets  arraylist of endorsements
     *
     * @return arraylist of endorsements that endorsed an original post
     */
    public ArrayList<Endorsement> getEndorsements() {
        return endorsements;
    }


    /** Gets arraylist of comments
     *
     * @return arraylist of comments that commented on an original post
     */
    public ArrayList<Comment> getComments() {
        return comments;
    }


    /** Adds an endorsement to the arraylist of endorsements in an original post
     *
     * @param endorsement endorsement to be added to the arraylist of endorsements of an original post
     */
    public void addEndorsement(Endorsement endorsement) {
        this.endorsements.add(endorsement);
    }


    /** Adds a comment to the arraylist of comments in an original post
     *
     * @param comment comment to be added to the arraylist of comments
     */
    public void addComment(Comment comment) {
        this.comments.add(comment);
    }
}
```

File: Post.java

```java
/**
 * Represents a Post, which is used as the parent class of any type of Post in the social media package, which is located in the social media package
 * A Post is made up of a message, an author and a post ID (pid)
 *
 * @author Jeroen Mijer, Alexander Robertson
 * @version 1.0
 * @since 1.0
 */
package socialmedia;
import java.io.Serializable;

public class Post implements Serializable {
    private int pid;
    private String message;
    private Account author;

    /**
     * Creates a Post
     *
     * @param message message of the Post
     * @param author author of the Post
     * @param pid post id of the Post
     */
    public Post (String message, Account author, Integer pid){
        this.pid = pid;
        this.message = message;
        this.author = author;
    }


    /** Gets the author of a post
     *
     * @return `Account that created the post (author)
     */
    public Account getAuthor() {
        return author;
    }

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

    /** Gets the post id (pid) of a post
     *
     * @return post id (pid) of a post
     */
    public int getPid() {
        return pid;
    }
}
```

File: SocialMedia.java

```java
package socialmedia;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Optional;

/**
 * SocialMedia is a compiling implementor of the SocialMediaPlatform interface.
 * SocialMedia is made up of an arraylist of all accounts created on the
 * platform and an arraylist of all posts made on the platform
 *
 * @author Jeroen Mijer
 * @author Alex Robertson
 * @version 1.0
 */
public class SocialMedia implements SocialMediaPlatform {

    // Public list of all accounts
    public transient ArrayList<Account> accounts;
    // public list of all posts
    public transient ArrayList<Post> posts;

    public SocialMedia() {
            /**
             * Creates a Social Media object Instantiates 2 empty arraylists of accounts and
             * posts because no account can be created or post can be posted until there is
             * a platform to be stored on
             *
             */
            this.accounts = new ArrayList<Account>();
            this.posts = new ArrayList<Post>();

            // create generic post with author "admin" to contain the generic error message
            Account genericAccount = new Account("admin", "", 1);
            OriginalPost genericPost = new OriginalPost(
                        "The original content was removed from the system and is no longer available.", genericAccount, 1);
            // add generic post and account to system arraylists
```

```
        this.accounts.add(genericAccount);
        this.posts.add(genericPost);
    }

    @Override
    public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
        // check if handle is valid
        // empty, more than 30 characters, has white spaces then is invalid
        if (handle.isEmpty() || handle.length() > 30 || handle.contains(" ")) {
            throw new InvalidHandleException("empty, OR more than 30 characters, OR has white spaces then is invalid");
        }

        // check if handle is unique
        for (Account account : accounts) {
            if (account.getHandle().equals(handle)) {
                throw new IllegalHandleException("That handle is not unique");
            }
        }

        // If handle input is valid
        // create account with descfield=""
        Integer uid = accounts.size() + 1;
        Account accountTemp = new Account(handle, "", uid);

        // add account to account list
        accounts.add(accountTemp);

        // return uid
        return accountTemp.getUID();

    }

    @Override
    public int createAccount(String handle, String description) throws IllegalHandleException, InvalidHandleException {
        // check if handle is valid
        // empty, more than 30 characters, has white spaces then is invalid
        if (handle.isEmpty() || handle.length() > 30 || handle.contains(" ")) {
            throw new InvalidHandleException(
                        "Handle empty, OR more than 30 characters, OR has white spaces then is invalid");
        }

        // check if handle is unique
        for (Account account : accounts) {
            if (account.getHandle().equals(handle)) {
                throw new IllegalHandleException("That handle is not unique");
            }
        }

        // If handle is valid
        // create account with descfield=description
        Integer uid = accounts.size() + 1;
        Account accountTemp = new Account(handle, description, uid);

        // add account to account list
        accounts.add(accountTemp);

        // return uid
        return accountTemp.getUID();
    }

    @Override
    public void removeAccount(int id) throws AccountIDNotRecognisedException {
        boolean found = false;
        for (int i = 0; i < accounts.size(); i++) {
            if (accounts.get(i).getUID() == id) {
                // for each post in the account delete the post using deletePost method
                for (Post post : accounts.get(i).getPosts()) {
                    try {
                        deletePost(post.getPid());
                    } catch (PostIDNotRecognisedException e) {
                        e.printStackTrace();
                    }
                }
                // remove account from the arraylist accounts
                accounts.remove(id - 1);
                found = true;
                break;
            }
        }
        if (!found) {
            throw new AccountIDNotRecognisedException("Account ID not recognised");
        }
    }

    @Override
    public void removeAccount(String handle) throws HandleNotRecognisedException {
        boolean found = false;
        for (int i = 0; i < accounts.size(); i++) {
            if (accounts.get(i).getHandle().equals(handle)) {

                for (Post post : accounts.get(i).getPosts()) {
                    try {
                        deletePost(post.getPid());
                    } catch (PostIDNotRecognisedException e) {
                        e.printStackTrace();
                    }
                }
                // remove account from the arraylist accounts
                accounts.remove(i);
                found = true;
                break;
            }
        }
        if (!found) {
            throw new HandleNotRecognisedException("Handle not recognised");
        }
    }

    @Override
    public void changeAccountHandle(String oldHandle, String newHandle)
            throws HandleNotRecognisedException, IllegalHandleException, InvalidHandleException {
        // Check if new handle is valid
        if (newHandle.isEmpty() || newHandle.length() > 30 || newHandle.contains(" ")) {
            throw new InvalidHandleException("empty, OR more than 30 characters, OR has white spaces then is invalid");
        }

        // check if new handle is unique
        for (Account account : accounts) {
            if (account.getHandle().equals(newHandle)) {
                throw new IllegalHandleException("That handle is not unique");
            }
        }

        // if input handle is valid
        // Change handle from oldHandle to newHandle
        boolean found = false;
        for (Account account : accounts) {
            if (account.getHandle().equals(oldHandle)) {
                account.setHandle(newHandle);
                found = true;
            }
        }
        // If input handle does not match any existing handle found throw handle not
        // recognised exception
        if (!found) {
            throw new HandleNotRecognisedException("Handle not recognised");
        }

    }

    @Override
    public void updateAccountDescription(String handle, String description) throws
HandleNotRecognisedException {
        boolean found = false;
        for (Account account : accounts) {
            if (account.getHandle().equals(handle)) {
                account.setDescField(description);
                found = true;
            }
        }
        // If input handle does not match any existing handle found throw handle not
        // recognised exception
        if (!found) {
            throw new HandleNotRecognisedException("Handle not recognised");
        }
    }

    @Override
    public String showAccount(String handle) throws HandleNotRecognisedException {
        boolean found = false;
        Account account = null;
        for (Account searchaccount : accounts) {
            if (searchaccount.getHandle().equals(handle)) {
                account = searchaccount;
                found = true;
            }
        }
        // If input handle does not match any existing handle found throw handle not
        // recognised exception
        if (!found) {
            throw new HandleNotRecognisedException("Handle not recognised");
        }

        Integer endorseCount = 0;
        // for all posts in account check the amount of endorsements and add them
to
```

```
        // endorseCount
        for (Post accPost : account.getPosts()) {
            // if post is an original post (in order to downcast)
            if (accPost instanceof OriginalPost) {
                endorseCount += ((OriginalPost) accPost).getEndorsements().size();
            }
            // else if post is Comment (in order to downcast)
            else if (accPost instanceof Comment) {
                endorseCount += ((Comment) accPost).getEndorsements().size();
            }

        }
        // returns the account in the desired format
        return String.format("""
                ID: %d
                Handle: %s
                Description: %s
                Post Count: %d
                Endorse Count: %d
                """, account.getUID(), account.getHandle(), account.getDescField(), account.getPosts().size(),
                endorseCount);
    }

    @Override
    public int createPost(String handle, String message) throws HandleNotRecognisedException, InvalidPostException {
        // verification of message
        if (message.length() <= 100 && !(message.isEmpty())) {
            // look through list of accounts
            for (Account account : accounts) {
                // find account with matching handle
                if (account.getHandle().equals(handle)) {
                    // Create a new original post
                    Integer pid = posts.size() + 1;
                    OriginalPost post = new OriginalPost(message, account, pid);
                    // Add the post to the list of posts in the account that created the post
                    account.addPost(post);
                    // Add the post to the list of posts in social media
                    this.posts.add(post);
                    return pid;
                }
            }
```

```java
            throw new HandleNotRecognisedException("Handle not recognised");
        } else {
            throw new InvalidPostException("Message is over 100 characters long OR is empty");
        }

    }

    private void addEndToPost(Post post, Endorsement endorsement) {
        if (post instanceof OriginalPost) {
            ((OriginalPost) post).addEndorsement(endorsement);
        } else {
            ((Comment) post).addEndorsement(endorsement);
        }
    }

    @Override
    public int endorsePost(String handle, int pid)
            throws HandleNotRecognisedException, PostIDNotRecognisedException, NotActionablePostException {
        // look through list of accounts

        for (Account account : accounts) {
            if (account.getHandle().equals(handle)) {
                // find post that wants to be endorsed using pid
                for (Post post : posts) {
                    // check if endorsing an endorsement
                    if (post.getPid() == pid) {
                        if (!(post instanceof Endorsement)) {
                            // Create the endorsement message in the desired format
                            String message = "EP@" + post.getAuthor().getHandle() + ": " + post.getMessage();
                            int endPID = posts.size() + 1;
                            Endorsement endorsement = new Endorsement(message, account, pid, endPID);
                            // Append endorsement to list of posts in social media
                            this.posts.add(endorsement);
                            // Add endorsement to list of posts of the account making the endorsement
                            account.addPost(endorsement);
                            addEndToPost(post, endorsement);
                            // check if endorsing an original post or a comment in order to downcast
                            return endorsement.getPid();
                        }
                        throw new NotActionablePostException("Cannot endorse an endorsement");
                    }
                }
                throw new PostIDNotRecognisedException("Post ID not recognised");
            }
        }
        throw new HandleNotRecognisedException("Handle is not recognised");
    }

    /**
     * This method is called exclusively from the commentPost method. It is to avoid
     * too many nested loops in one method This is mainly due to needing to downcast
     * from a post to either a comment or an originalpost
     *
     * @param message message of the comment
     * @param account account that is commenting on a post
     * @param pid     post ID of the post that is being commented on
     * @param post    post that is being commented on
     * @return post ID of the comment
     * @throws NotActionablePostException
     */
    private int cognitiveComplexityReducer(String message, Account account, int pid, Post post)
            throws NotActionablePostException {
        if (post instanceof Endorsement) {
            throw new NotActionablePostException("Cannot comment on an endorsement");
        }
        int comPID = posts.size() + 1;
        Comment comment = new Comment(message, account, pid, comPID);
        // have to add endorsement to list of posts, account and make it a child of
        // other post
        this.posts.add(comment);
        account.addPost(comment);
        // check if endorsing an original post or a comment in order to downcast
        // Add comment to arraylist of comments of parent post
        if (post instanceof OriginalPost) {
            ((OriginalPost) post).addComment(comment);
        }
        if (post instanceof Comment) {
            ((Comment) post).addComment(comment);
        }
        return comment.getPid();

    }
```

```java

    @Override
    public int commentPost(String handle, int pid, String message) throws HandleNotRecognisedException,
            PostIDNotRecognisedException, NotActionablePostException, InvalidPostException {
        // test if message is valid
        if (!(message.length() <= 100 && !message.isEmpty())) {
            throw new InvalidPostException("Message of post is greater than 100 characters OR is empty");
        }

        boolean found = false;
        Account accountCommentor = null;
        for (Account account : accounts) {
            if (account.getHandle().equals(handle)) {
                accountCommentor = account;
                found = true;
            }
        }
        // If input handle does not match any existing handle found throw handle not
        // recognised exception
        if (!(found)) {
            throw new HandleNotRecognisedException("Handle not recognised");
        }

        for (Post post : posts) {
            if (post.getPid() == pid) {
                return cognitiveComplexityReducer(message, accountCommentor, pid, post);
            }
        }
        throw new PostIDNotRecognisedException("Post ID not recognised");

    }

    @Override
    public void deletePost(int pid) throws PostIDNotRecognisedException {
        boolean found = false;
        for (Post post : posts) {
            if (post.getPid() == pid) {
                // Check what kind of post is being deleted and call respective function
                if (post instanceof OriginalPost) {
                    deleteOriginalPost((OriginalPost) post);
                    found = true;
                    break;
                } else if (post instanceof Comment) {
                    deleteComment((Comment) post);
                    found = true;
                    break;
                } else if (post instanceof Endorsement) {
                    deleteEndorsement((Endorsement) post);
                    found = true;
                    break;
                    // all types are stored in socialmedia.posts
                    // account has an arraylist of all posts
                }
            }
        }
        if (!found) {
            throw new PostIDNotRecognisedException("Post ID not recognised");
        }

    }

    /**
     * Deletes an original post
     *
     * @param post post that is being deleted
     */
    private void deleteOriginalPost(OriginalPost post) {
        for (int i = 0; i < post.getEndorsements().size(); i++) {
            deleteEndorsement(post.getEndorsements().get(i));
        }

        OriginalPost genPost = (OriginalPost) posts.get(0);
        for (Comment comment : post.getComments()) {
            /// set the post pointer of any comments that commented on this post to 1
            // a post pointer of 1 corresponds to the generic deleted message
            comment.setPostPointer(1);
            // add comment to generic post
            genPost.addComment(comment);
        }
        // remove the original post from the account that created it
        Account author = post.getAuthor();
        author.removePost(post);
        // remove the original post from the arraylist of posts in social media
        this.posts.remove(post);

    }

    /**
```

```java
     * Deletes a comment
     *
     * @param comment comment that is being deleted
     */
    private void deleteComment(Comment comment) {
        OriginalPost genPost = (OriginalPost) posts.get(0);
        for (Comment commentChild : comment.getComments()) {
            // set the post pointer of any comments that commented on this post to 1
            // a post pointer of 1 corresponds to the generic message
            commentChild.setPostPointer(1);
            commentChild.setPostPointer(1);
            // add comment to genenric post
            genPost.addComment(comment);
        }
        //remove the comment from the account that created it
        Account author = comment.getAuthor();
        author.removePost(comment);
        //remove the comment from the arraylist of posts in social media
        this.posts.remove(comment);

        //remove comment from the arraylist of comments of the post that it commented on
        for (Post post : posts) {
            if (comment.getPostPointer() == post.getPid()) {
                if (post instanceof OriginalPost) {
                    ((OriginalPost) post).getComments().remove(comment);
                    break;
                } else if (post instanceof Comment) {
                    ((Comment) post).getComments().remove(comment);
                    break;
                }
            }
        }
    }

    /**
     * Deletes an endorsement
     * @param endorsement endorsement that is being deleted
     */
    private void deleteEndorsement(Endorsement endorsement) {

        //remove the endorsement from the account that created it
        Account author = endorsement.getAuthor();
        author.removePost(endorsement);

        //remove the endorsement from the arraylist of posts in social media
        this.posts.remove(endorsement);

        //remove endorsement from the arraylist of comments of the post that it commented on
        for (Post post : posts) {
            if (endorsement.getPostPointer() == post.getPid()) {
                if (post instanceof OriginalPost) {
                    ((OriginalPost) post).getEndorsements().remove(endorsement);
                } else if (post instanceof Comment) {
                    ((Comment) post).getEndorsements().remove(endorsement);
                }
            }
        }

    }

    @Override
    public String showIndividualPost(int pid) throws PostIDNotRecognisedException {
        for (Post post : posts) {
            if (post.getPid() == pid) {
                String id = "ID: " + pid + "\n";
                String account = "Account: " + posts.get(pid - 1).getAuthor() + "\n";
                int numberOfEndorsements;
                int numberOfComments;
                //If post is an endorsement it will have no endorsements or comments
                if (post instanceof Endorsement) {
                    numberOfEndorsements = 0;
                    numberOfComments = 0;
                } else if (post instanceof OriginalPost) {
                    numberOfEndorsements = ((OriginalPost) post).getEndorsements().size();
                    numberOfComments = ((OriginalPost) post).getComments().size();
                } else {
                    numberOfEndorsements = ((Comment) post).getEndorsements().size();
                    numberOfComments = ((Comment) post).getComments().size();
                }
                String number = "No. endorsements: " + numberOfEndorsements + " | No. Comments: " + numberOfComments
                        + "\n";
				//combine elements of individual post into one string
				// formatting was done in the form of adding new line characters where relevant
				return id + account + number + post.getMessage();
			}
		}
		throw new PostIDNotRecognisedException("Post ID not recognised");
	}

	/** Appends each individual post to its parent post in order to show post children details
	 * Deals with formatting inside the function
	 * Is recursive and calls itself
	* @param indentation the indentation that the post should be displayed at
	* @param post the parent post that the child post is being appended to
	* @return A post and the details of all its children
	* @throws PostIDNotRecognisedException
	*/
	private StringBuilder recursionFunc(Integer indentation, Comment post) throws PostIDNotRecognisedException {
		StringBuilder formatCom = new StringBuilder();
		// increment indentation level
		indentation++;
		// get post message
		String message = showIndividualPost(post.getPid());
		// indent accordingly
		message = message.indent(indentation * 4);
		// change first indent to match format
		message = message.replaceFirst("    ID", "| > ID");
		// if length of arraylist comments is greater than 0, add an extra | underneath
		if (!post.getComments().isEmpty()) {
			String spaces = " ";
			message += spaces.repeat(indentation * 4) + "|\n";
		}

		// append message to stringbuilder
		formatCom.append(message);

		// repeat process for each comment
		for (Comment comment : post.getComments()) {
			formatCom.append(recursionFunc(indentation, comment));// recursion
		}
		return formatCom;
	}

	@Override
```

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

        Optional<Post> postOpt;// an optional can be null as well
        postOpt = posts.stream().filter(p -> p.getPid() == id).findFirst();// finds the first (and only) instance a post
                                                                            // with matching pid
        if (!postOpt.isPresent()) {// if it was found
            throw new PostIDNotRecognisedException();
        }
        Post post = postOpt.get();// actually gets the post from the 'optional'
        if (post instanceof Endorsement) {// if endorsement
            throw new NotActionablePostException();
        }

        StringBuilder format = new StringBuilder();
        format.append(showIndividualPost(id));// print original post/comment

        // idc if comment or original post, go through comments and call recursiveFunc
        // to print tree
        if (post instanceof OriginalPost) {
            if (!((OriginalPost) post).getComments().isEmpty()) {
                format.append("\n|\n");
            }
            for (Comment comment : ((OriginalPost) post).getComments()) {
                format.append(recursionFunc(0, comment));
            }
        } else {
            if (!((Comment) post).getComments().isEmpty()) {
                format.append("\n|\n");
            }
            for (Comment comment : ((Comment) post).getComments()) {
                format.append(recursionFunc(0, comment));
            }
        }

        return format;
    }

    @Override
    public int getNumberOfAccounts() {
        return accounts.size();
    }

    @Override
    public int getTotalOriginalPosts() {
        //If a post is an instance of an original post increase the counter by 1
        long counter = posts.stream().filter(p -> p instanceof OriginalPost).count();
        return (int) counter;
    }

    @Override
    public int getTotalEndorsmentPosts() {
        // DO NOT alter typo in method declaration or the tests wont run properly
        //If a post is an instance of an endorsement increase the counter by 1
        long counter = posts.stream().filter(p -> p instanceof Endorsement).count();
        return (int) counter;
    }

    @Override
    public int getTotalCommentPosts() {
        //If a post is an instance of a comment increase the counter by 1
        long counter = posts.stream().filter(p -> p instanceof Comment).count();
        return (int) counter;
    }

    @Override
    public int getMostEndorsedPost() {
        //If there are no posts with endorsements then will return an invalid post ID of -1
        Integer maxEnd = -1;
        Integer maxPID = -1;
        // For each post check the number of endorsements
        // If the post has a larger number than the previous maximum number of endorsements update the counter and post ID for most endorsed post accordingly
        for (Post post : posts) {
            if (post instanceof OriginalPost) {
                long counter = ((OriginalPost) post).getEndorsements().stream().count();
                if ((int) counter > maxEnd) {
                    maxEnd = (int) counter;
                    maxPID = post.getPid();
                }
            }
            if (post instanceof Comment) {
                long counter = ((Comment) post).getEndorsements().stream().count();
                if ((int) counter > maxEnd) {
                    maxEnd = (int) counter;
                    maxPID = post.getPid();
                }
            }

        }
        return maxPID;
    }

    @Override
    public int getMostEndorsedAccount() {
        //If there are no accounts containing posts with endorsements then will return an invalid post ID of -1
        Integer maxEnd = -1;
        Integer maxUID = -1;
        for (Account account : accounts) {
            //for each account set its endorsement counter to 0
            long counter = 0L;
            // iterate through each post in an account and add the number of endorsements to the account counter
            for (Post post : account.getPosts()) {
                if (post instanceof OriginalPost) {
                    counter += ((OriginalPost) post).getEndorsements().stream().count();
                }
                if (post instanceof Comment) {
                    counter += ((Comment) post).getEndorsements().stream().count();
                }
            }
            // if the account counter is larger than the largest previous counter, update the counter and post ID to the new most endorsed account
            if ((int) counter > maxEnd) {
                maxEnd = (int) counter;
                maxUID = account.getUID();
            }

        }
        return maxUID;
    }

    @Override
    public void erasePlatform() {
        // Method empties this SocialMediaPlatform of its contents and resets all
        // internal counters.
        // set lists to empty
        this.accounts.clear();
        this.posts.clear();

    }

```

```java
	@Override
	public void savePlatform(String filename) throws IOException {
		// Prepend platform content to file filename
		try (FileOutputStream file = new FileOutputStream(filename, false);
				ObjectOutput stream = new ObjectOutputStream(file)) {
			Integer length = accounts.size();//

			stream.writeObject(length);
			for (Account account : accounts) {
				stream.writeObject(account);
			}

			length = posts.size();

			stream.writeObject(length);
			for (Post post : posts) {
				stream.writeObject(post);

			}

			stream.flush();

		}

	}

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

		try (FileInputStream file = new FileInputStream(filename);
				ObjectInputStream stream = new ObjectInputStream(file);) {

			Integer length = (Integer) stream.readObject();
			for (int i = 0; i < length; i++) {
				accounts.add((Account) stream.readObject());
			}

			length = (Integer) stream.readObject();
			for (int j = 0; j < length; j++) {
				posts.add((Post) stream.readObject());
			}
		}
	}
}
```