1 Account.java

```java
package socialmedia;

//Imports
import java.util.ArrayList; //used to store a dynamic list of objects
import java.util.Iterator; //used to iterate through ArrayList objects
import java.io.Serializable; //allows the state of the platform to be saved as a byte stream

/**
 * The class that contains all of the user account objects. These consist of a unique ID, a handle, a
     description and an ArrayList of posts made by this account. The class has the attribute
     NO_OF_ACCOUNTS, which is used to create IDs
 *
 * @author Jack Skinner, Eleanor Forrest
 */

public class Account implements Serializable {
    //Attributes
    private static int NO_OF_ACCOUNTS = 0;
    private int id;
    private String handle;
    private String description;
    private int noOfEndorsements = 0;
    private ArrayList<Post> posts = new ArrayList<>();

    /**
     * The first constructor for the account does not set a description and instead intitalises it to an
         empty string
     *
     *  @param handle - String: the handle of the account that the user wishes to create
     *
     * */
    public Account(String handle) {
        this.handle = handle;
        this.id = ++NO_OF_ACCOUNTS;
        this.description = "";
    }

    /**
     * The second constructor for the account takes the description as a parameter as well as the handle
     *
     *  @param handle - String: the handle of the account that the user wishes to create
     *  @param description - String: the description for the account
     *
     * */
    public Account(String handle, String description) {
        this.handle = handle;
        this.id = ++NO_OF_ACCOUNTS;
        this.description = description;
    }

    /**
     * Create a Post object associated with this account, adding it to the list of posts
     *
     * @param message - String: the message that the post should contain
     * @return id - int: the ID generated by NO_OF_POSTS which is automatically assigned and incremented
     *
     * */
    public int makePost(String message) {
        Post p = new Post(message);
        posts.add(p);
        return p.getId();
    }

    /**
     * Create a comment object, given the ID of the original post and a message
     *
     * @param originalId - int: The ID of the original post this comment references
     * @param message - String: The message that the comment will have
     * @return id - The ID of this comment, generated by NO_OF_POSTS which is automatically assigned and
         incremented
     */
    public int makeComment(int originalId, String message) {
        Comment c = new Comment(message, originalId);
        posts.add(c);
        return c.getId();
    }

    /**
     * Create an endorsement, given the ID of the original post, the handle of the original poster, and the
         original post's message.
     * The message is formatted to show that it is an endorsement
     *
     * @param originalId - int : The ID of the original post
     * @param originalHandle - String: The handle of the original poster
     * @param orignalMessage - String: The message of the original post
     * @return id - int: the ID of this endorsement, generated by NO_OF_POSTS which is automatically
         assigned and incremented
     */
    public int makeEndorsement(int originalId, String originalHandle, String orignalMessage) {
        String message = ("EP@" + originalHandle + ": " + orignalMessage);
        Endorsement e = new Endorsement(message, originalId);
        posts.add(e);
        return e.getId();
    }

    /**
     * void method, used whenever an endorsement post referring to a post on this account is created.
     * increments the noOfEndorsements attribute of the account object
     */
    public void endorsed() {
        noOfEndorsements += 1;
    }

    /**
     * void method, used whenever an endorsement post referring to a post on this account is deleted
     * decrements the noOfEndorsements attribute of the account object
     */
```

```java
    public void unendorsed(){
        noOfEndorsements -= 1;
    }

    /**
     * check if an account has a specific post before getting it. This avoids exceptions raised trying to
         access a post that doesn't exist
     * @param id - int: the ID of the post that is being looked for
     * @return boolean: whether or not this ID is linked to a post in the posts ArrayList
     */
    public boolean hasPost(int id) {
        for (Post p : posts){
            if (p.getId() == id){
                return true;
            }
        }
        return false;
    }

    /**
     * Retrieve a requested post based on its ID, usually called after hasPost()
     *
     * @param id - int: the ID of the post that is being grabbed
     * @return p - Post: the post that is linked to that ID
     * @throws PostIDNotRecognisedException - thrown if the ID does not match a post from this account
     */
    public Post getPost(int id) throws PostIDNotRecognisedException{
        for (Post p : posts){
            if (p.getId() == id){
                return p;
            }
        }
        throw new PostIDNotRecognisedException();
    }

    /**
     * This method deletes a post from the posts ArrayList, removing its link to the account that created it
     *
     * @param id - The ID of the post that is being deleted from this account
     * @throws PostIDNotRecognisedException - thrown if the ID is not linked to a post from this account
     */
    public void deletePost(int id) throws PostIDNotRecognisedException{
        Iterator<Post> itr = posts.iterator();
      while (itr.hasNext()) {
         Post p = (Post)itr.next();
            if (p.getId() == id){
                p.setMessage("The original content was removed from the system and is no longer available.");
                p.setPostType("DeletedPost");
                itr.remove();
                return;
            }

            }
        throw new PostIDNotRecognisedException();
    }

    /**
     * This method removes every post in the posts ArrayList. This is called before an account is deleted,
         to remove all of its associated posts
     */
    public void deleteAllPosts() {
        Iterator<Post> itr = posts.iterator();
      while (itr.hasNext()) {
         Post p = (Post)itr.next();
            p.setMessage("The original content was removed from the system and is no longer available.");
            p.setPostType("DeletedPost");
            itr.remove();
        }
    }

    /**
     * void method, used when the platform is reset. Sets the counter for NO_OF_ACCOUNTS to zero, so the IDs
         start incrementing from 1 again
     */
    public static void reset(){
        NO_OF_ACCOUNTS = 0;
    }

    /**
     * getter, returnes the private static attribute NO_OF_ACCOUTNS
     * @return NO_OF_ACCOUNTS - int: the number of accounts ever created, used to increment the ID of new
         accounts
     */
    public static int getNO_OF_ACCOUNTS() {
        return NO_OF_ACCOUNTS;
    }

    /**
     * getter, returns the handle
     * @return handle - String: the handle of the account
     */
    public String getHandle() {
        return handle;
    }

    /**
     * getter, returns the ID
     * @return id - int: the ID of the account
     */
    public int getId() {
        return id;
    }

    /**
     * getter, returns the description of the account
     * @return description - String: the description of the account created
     */
    public String getDescription() {
        return description;
    }

    /**
     * setter, updates the handle of the account object
     * @param handle - String: the new handle that will be assigned to the account
     */
    public void setHandle(String handle) {
        this.handle = handle;
    }

    /**
     * setter, updates the description of the account object
     * @param description - String: the new description that will be assigned to the account
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     * getter - gets the size of the posts ArrayList, the number of not deleted posts created by this account
     * @return posts.size - int: the size of the posts ArrayList
     */
    public int getNoOfPosts() {
        return posts.size();
    }

    /**
     * getter - returns the attribute numberOfEndorsements
     * @return noOfEndorsements - int: The number of Endorsements linked to this account
     */
    public int getNoOfEndorsements() {
        return noOfEndorsements;
    }
    /**
     * getter - loops through the posts ArrayList, to retrieve any Comment objects and add them to a new
         ArrayList, which is used in showPostChildrenDetails()
     * @return comments - ArrayList<Comment> : a list of comments created by this account
     */
    public ArrayList<Comment> getComments() {
        ArrayList<Comment> comments = new ArrayList<>();
        for (Post p : posts) {
            if (p.getPostType().equals("CommentPost")) {
                Comment c = (Comment)p;
                comments.add(c);
            }
        }
        return comments;
    }

    /**
     * getter, returns the ArrayList containing this account's posts
     * @return posts - ArrayList<Post> : an ArrayList of all of the account's posts
     */
    public ArrayList<Post> getPosts(){
        return posts;
    }

    /**
     * void method, used to set the number of accounts when loading the platform from a file. This ensures the IDs of newly created accounts are correct
     * @param no - int: the value of NO_OF_ACCOUNTS that was saved when savePlatform() was invoked
     */
    public static void setNO_OF_ACOUNTS(Integer no) {
        NO_OF_ACCOUNTS = no;
    }

}
```

2 Post.java

```java
package socialmedia;

//Imports
import java.io.Serializable; //allows the state of the platform to be saved as a byte stream

/**
 * The class that contains Post objects, which is a superclass for Endorsement and Comment objects. Objects have an ID, message, type, number of endorsements and number of comments. The class has the attribute NO_OF_POSTS, which is used to create IDs
 *
 * @author Jack Skinner, Eleanor Forrest
 */

public class Post implements Serializable{
    //Attributes
    private static int NO_OF_POSTS = 0; //NO_OF_POSTS is the total number of posts created rather than the number of posts currently in the platform, used for generating unique IDs
    protected int postId;
    protected String message;
    protected String postType = "OriginalPost";
    protected int numberOfEndorsements = 0;
    protected int numberOfComments = 0;

    /**
     * The constructor for the post class
     * @param message - String: The message that the post will contain
     */
    public Post(String message) {
        this.message = message;
        //The ID of the post is set to how many posts were created before it
        postId = ++NO_OF_POSTS;
    }

    /**
     * void method, used when a post is endorsed to increment its number of endorsements
     */
    public void addEndorsement(){
        numberOfEndorsements += 1;
    }

    /**
     * void method, used when an endorsement is deleted to decrement its number of endorsements
     */
    public void removeEndorsement(){
        numberOfEndorsements -= 1;
    }

    /**
     * void method, used when a post receives a comment to increment its number of comments
     */
    public void addComment(){
        numberOfComments += 1;
    }

    /**
     * void method, used when a post's comment is deleted to decrement its number of comments
     */
    public void removeComment(){
        numberOfComments -= 1;
    }

    /**
     * void method, called when the platform is erased. Resets NO_OF_POSTS to 0 so IDs start from 1
     */
    public static void reset(){
        NO_OF_POSTS = 0;
    }

    /**
     * getter, returns the total number of posts created on the platform
     * @return NO_OF_POSTS - int: number of posts ever created
     */
    public static int getNO_OF_POSTS() {
        return NO_OF_POSTS;
    }

    /**
     * getter, returns a post's message
     * @return message - String: the message of the post
     */
    public String getMesssage() {
        return message;
    }

    /**
     * getter, returns a post's ID
     * @return postId - int: the ID of the post
     */
    public int getId(){
        return postId;
    }

    /**
     * getter, returns a post's type (Original, Comment, Endorsement or Deleted)
     * @return postType - String: the type of the post
```

```java
     */
    public String getPostType(){
        return postType;
    }

    /**
     * getter, returns the number of endorsements a post has received
     * @return numberOfEndorsements - int: the number of endorsements received by a post
     */
    public int getNumberOfEndorsements(){
        return numberOfEndorsements;
    }

    /**
     * getter, returns the number of comments a post has received
     * @return numberOfComments - int: the number of comments received by a post
     */
    public int getNumberOfComments(){
        return numberOfComments;
    }

    /**
     * setter, sets the message of a post to the parameter
     * @param message - String: the new message the post will have
     *
     */
    public void setMessage(String message){
        this.message = message;
    }

    /**
     * setter, sets the post type to a new type, used when a comment is deleted and stored in deletedComments
     * @param postType - String: the type the post should be updated to
     */
    public void setPostType(String postType){
        this.postType = postType;
    }

    /**
     * setter, when the platform is loaded from a file, NO_OF_POSTS is saved so IDS are incremented correctly
     * @param no - Integer: the number of posts that the platform had when saved
     */
    public static void setNO_OF_POSTS(Integer no){
        NO_OF_POSTS = no;
    }
}
```

3 Comment.java

```java
package socialmedia;

/**
 * This class extends the Post superclass, used for comments. Has an additional attrubte originalPostId
     which refers to the post being commented on
 *
 * @author Jack Skinner, Eleanor Forrest
 */

public class Comment extends Post{
    private int originalPostID;

    /**
     * constructor, creates a comment, given a message and orignalPostID
     * @param message - String: the message that the comment will have
     * @param originalPostID int: The ID of the post this comment is commenting on
     */
    public Comment(String message, int originalPostID) {
    super(message);
    this.originalPostID = originalPostID;
    //as in Endorsement, the post type is set to OriginalPost in the superclass, this corrects it
    this.postType = "CommentPost";
    }

    /**
     * getter, returns the attribute originalPostId
     * @return originalPostId - int: the ID of the post this comment has commented on
     */
    public int getOriginalPostID() {
        return originalPostID;
    }

}
```

4 Endorsement.java

```java
package socialmedia;

/**
 * This class extends the Post superclass, used for endorsements. Has an additional attrubte originalPostId
     which refers to the post being endorsed
 *
 * @author Jack Skinner, Eleanor Forrest
 */
public class Endorsement extends Post{

    private int originalPostId;

    /**
     * constructor, creates an endorsement, given a message and orignalPostID
     * @param message - String: the message this endorsement will have, formatted correctly in
         makeEndorsement()
     * @param originalPostId - int: the ID of the post that this endorsement is endorsing
     */
    public Endorsement(String message, int originalPostId) {
        super(message);
        this.originalPostId = originalPostId;
        //the post type is set to OriginalPost in the superclass, this corrects it
        this.postType = "EndorsementPost";

    }

    /**
     * getter, returns the attribute originalPostId
     * @return originalPostId - int: the ID of the post this endorsement is endorsing
     */
    public int getOriginalPostId() {
        return originalPostId;
    }

}
```

5 SocialMedia.java

```java
package socialmedia;

//Imports
import java.io.IOException; //thrown if there is an issue saving or loading the file
import java.util.Scanner; //used when generating the string of posts for showPostChildrenDetails()
import java.util.ArrayList; //used to store a dynamic list of objects
import java.util.Iterator; //used to iterate through ArrayList objects
//The following imports are used to handle saving and loading the platform as a byte stream
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.FileInputStream;

/**
 * SocialMedia is a functioning implementation of the SocialMediaPlatform interface providing the backend
     for this project
 *
 * @author Jack Skinner and Eleanor Forrest
 *
 */
public class SocialMedia implements SocialMediaPlatform {
   private ArrayList<Account> accounts = new ArrayList<>(); //contains all the Account objects that exist
       in the platform
   private ArrayList<Comment> deletedComments = new ArrayList<>(); //contains any Comment objects that have
       been deleted, so that any successive comments can still refer to them, thus preventing them from
       being removed by the garbage collector


   /**
    * returns an account given it's handle. Throws HandleNotRecognisedException if the handle isn't saved
        in accounts
    * @param handle - String: The handle of the account that is being searched for
    * @return a - Account: The account with said handle
    * @throws HandleNotRecognisedException - Thrown if the handle is not found
    */
   private Account returnAccount(String handle) throws HandleNotRecognisedException {
      //given an account handle, return the account object
      for(Account a : accounts) {
         if (a.getHandle().equals(handle)) {
            return a;
         }
      }
      throw new HandleNotRecognisedException(); //if the account with this handle doesn't exist
   }

   @Override
   public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
      //check if the account handle is valid
      if ((handle.isEmpty()) || (handle.length() > 30) || (handle.contains(" "))) {
         throw new InvalidHandleException();
      }
      //search the accounts ArrayList to see if the handle is already in use
      for (Account a : accounts) {
         if (a.getHandle().equals(handle)) {
            throw new IllegalHandleException();
         }
      }
      //if all checks are passed, create a new account with the verified handle
      Account newAccount = new Account(handle);
      accounts.add(newAccount);
      //return the ID of the new account
      return newAccount.getId();
   }

   @Override
   public int createAccount(String handle, String description) throws IllegalHandleException,
       InvalidHandleException {
      //call the original createAccount() method with only the handle
      int id = createAccount(handle);
      //loop through each account in accounts, to find the account that was just created using the ID
      for (Account a : accounts) {
         if (a.getId() == id) {
            //set the description of this account to the description given in the input
            a.setDescription(description);
         }
      }
      return id;
   }

   @Override
   public void removeAccount(int id) throws AccountIDNotRecognisedException {
      //Iterator is used to iterate through the accounts ArrayList and delete items without index errors
      Iterator<Account> itr = accounts.iterator();
      while (itr.hasNext()) {
         //if the current account has the ID we are looking to delete, start deleting
         Account a = (Account)itr.next();
         if (a.getId() == id){
            //go through each post owned by this account and delete it
            ArrayList<Post> posts = a.getPosts();
            while(!posts.isEmpty()){
               Post p = posts.get(0);
               //deletePost() throws PostIDNotRecognisedException, this will never be raised however we need
                   to handle it
               try{
                  deletePost(p.getId());
               } catch (PostIDNotRecognisedException e){

               }

            }
            //set the account to null, and remove it from the iterator. It will be removed from the heap by the garbage collector
            a = null;
            itr.remove();
            return;
         }
      }
      //throw AccountIDNotRecognisedException if no account is found with the matching ID
      throw new AccountIDNotRecognisedException();

   }

   @Override
   public void removeAccount(String handle) throws HandleNotRecognisedException {
      //get the account to be removed based on its handle. We already have a function to do this and so do not need to use an iterator
      Account account = returnAccount(handle);
      //remove the posts associated with the account similarly
      ArrayList<Post> posts = account.getPosts();
      while(!posts.isEmpty()){
         Post p = posts.get(0);
         try{
            deletePost(p.getId());
         } catch (PostIDNotRecognisedException e){

         }

      }
      //remove the account from accounts and set it to null
      accounts.remove(account);
      account = null;
   }

   @Override
   public void changeAccountHandle(String oldHandle, String newHandle)
         throws HandleNotRecognisedException, IllegalHandleException, InvalidHandleException {
      //find the account to change the handle of
      Account a = returnAccount((oldHandle));
      //check if the new handle is already in use by looping through each account in accounts
      for (Account b : accounts) {
         if (b.getHandle().equals(newHandle)) {
            throw new IllegalHandleException();
         }
      }
      //check that the new handle is valid
      if ((newHandle.isEmpty()) || (newHandle.length() > 30) || (newHandle.contains(" "))) {
         throw new InvalidHandleException();
      }
      //change the handle
      a.setHandle(newHandle);
   }

   @Override
   public void updateAccountDescription(String handle, String description) throws
       HandleNotRecognisedException {
      //find the account to be edited and set its description to the new description
      Account a = returnAccount(handle);
      a.setDescription(description);
   }

   @Override
   public String showAccount(String handle) throws HandleNotRecognisedException {
      //generate a string containing information about the requested account
      Account a = returnAccount(handle);
      String accountOut = "";
      accountOut += ("ID: " + Integer.toString(a.getId()) + " \n");
      accountOut += ("Handle: " + a.getHandle() + " \n");
      accountOut += ("Description: " + a.getDescription() + " \n");
      accountOut += ("Post count: " + Integer.toString(a.getNoOfPosts()) + " \n");
      accountOut += ("Endorse count: " + Integer.toString(a.getNoOfEndorsements()) + " \n");
      return accountOut;
   }

   @Override
   public int createPost(String handle, String message) throws HandleNotRecognisedException,
       InvalidPostException {
      Account a = returnAccount(handle);
      //check that the post is valid
      if ((message.isEmpty()) || (message.length() > 100)) {
         throw new InvalidPostException();
      }
      //create the post
      int postId = a.makePost(message);
      return postId;
   }

   @Override
   public int endorsePost(String handle, int id)
         throws HandleNotRecognisedException, PostIDNotRecognisedException, NotActionablePostException {
      Account endorsing = returnAccount(handle); //endorsing is the account which is endorsing a post
      //find the post to be endorsed, if it isn't found PostIDNotRecognisedException is thrown
      for (Account endorsed : accounts) { //endorsed is the account which owns the post that the endorsing
          account wishes to endorse
         if (endorsed.hasPost(id)) {
            //get the post to be endorsed
            Post originalPost = endorsed.getPost(id);
            //check that this post is a post that can be endorsed, else throw NotActionablePostException
            if (originalPost.getPostType().equals("EndorsementPost") ||
                originalPost.getPostType().equals("DeletedPost")) {
               throw new NotActionablePostException();
            }
            //create the new endorsement post
            int newID = endorsing.makeEndorsement(id, endorsed.getHandle(), originalPost.getMesssage());
            //increment the number of endorsed posts the owner of this post has and the number of
```

```
                endorsements on the post
            endorsed.endorsed();
            originalPost.addEndorsement();
            return newID;
         }
      }
      throw new PostIDNotRecognisedException();
   }

   @Override
   public int commentPost(String handle, int id, String message) throws HandleNotRecognisedException,
         PostIDNotRecognisedException, NotActionablePostException, InvalidPostException {
      Account commenting = returnAccount(handle); //commenting is the account making a comment
      //find the post that this account wants to make a comment on
      for (Account commented : accounts) { //commented is the account which owns the post being commented on
         if (commented.hasPost(id)) {
            Post originalPost = commented.getPost(id);
            //check that this post is a post that can be commented on
            if (originalPost.getPostType().equals("EndorsementPost") ||
                originalPost.getPostType().equals("DeletedPost")) {
               throw new NotActionablePostException();
            }
            //check that the comment is valid
            if ((message.isEmpty()) || (message.length() > 100)) {
               throw new InvalidPostException();
            }
            //create the new comment post
            int newId = commenting.makeComment(id, message);
            //increment the number of comments that the original post has
            originalPost.addComment();
            return newId;
         }
      }
      throw new PostIDNotRecognisedException();
   }

   @Override
   public void deletePost(int id) throws PostIDNotRecognisedException {
      //find the post to be deleted, throwing PostIDNotRecognisedException if it isn't found
      for (Account a : accounts) {
         if (a.hasPost(id)) {
            Post p = a.getPost(id);
            //deal with if the post is an endorsement post - decrement the number of endorsements the post
                and the account which was endorsed have
            if (p.getPostType().equals("EndorsementPost")){
               //to find the original post p must be downcasted into an endorsement object
                   Endorsement e = (Endorsement)p;
                   int originalPostId = e.getOriginalPostId();
               //find the account with the original post
                   for(Account a2 : accounts){
                 if (a2.hasPost(originalPostId)){
                    a2.getPost(originalPostId).removeEndorsement();
                    a2.unendorsed();
                 }
               }
```

```java
                }
                //deal with if the post is a comment post - decrement the number of comments on the original post
                if (p.getPostType().equals("CommentPost")){
                    //to find the original post p must be downcasted into a comment object
                    Comment c = (Comment)p;
                    int originalPostId2 = c.getOriginalPostID();
                    for(Account a3 : accounts){
                        if (a3.hasPost(originalPostId2)){
                            a3.getPost(originalPostId2).removeComment();
                        }
                    }
                }
                //deal with any comments that refer to the post being deleted - if there are any, this post must be added to the deletedComments ArrayList so that when showPostChildrenDetails() is called the children comments refer to a post with a dummy message
                for (Account a4: accounts){
                    for (Comment c2 : a4.getComments()){
                        if (c2.getOriginalPostID() == id && p.getPostType().equals("CommentPost")){
                            deletedComments.add((Comment)p);
                        }
                    }
                }
                //delete the post
                a.deletePost(id);
                return;
            }
        }
        throw new PostIDNotRecognisedException();

    }

    @Override
    public String showIndividualPost(int id) throws PostIDNotRecognisedException {
        //find the requested post
        for (Account a : accounts){
            if (a.hasPost(id)){
                //generate a string containing information about the post
                Post post = a.getPost(id);
                String postDetails = "";
                postDetails += "ID: "+Integer.toString(id)+" \n";
                postDetails += "Account: "+a.getHandle()+" \n";
                postDetails += "No. endorsements: " + Integer.toString(post.getNumberOfEndorsements()) +" | No. comments: " + Integer.toString(post.getNumberOfComments()) + " \n";
                postDetails += post.getMesssage() +"\n";
                //the string is now formatted appropriatley, and returned
                return postDetails;
            }
        }

        throw new PostIDNotRecognisedException();
    }

    public StringBuilder showPostChildrenDetails(int id)
            throws PostIDNotRecognisedException, NotActionablePostException {
        // loop through all accounts
        for (Account a: accounts){
            //if the account has a post with the matching id, check if the post is and original post or a comment. If not, throw NotActionablePostException
            if (a.hasPost(id)){
                if (a.getPost(id).getPostType().equals("EndorsementPost") || a.getPost(id).getPostType().equals("DeletedPost")){
                    throw new NotActionablePostException();
                }
                //create a StringBuilder to contain the eventual string to be returned, and append the string returned from calling showIndividualPost() on the parent post
                StringBuilder postChildrenDetails = new StringBuilder();
                postChildrenDetails.append(showIndividualPost(id));
                //enter recursivePost() to build the string, starting with a depth of 0
                recursivePost(a.getPost(id), 0, postChildrenDetails);
                return postChildrenDetails;
            }
        }
        throw new PostIDNotRecognisedException();
    }

    /**
     * Recursive solution to building the children details. Displays the thread properly formatted with the |'s and indents
     * Each child calls this method with all of their own children posts, until a post has no comments, where the base case is met
     * @param post - Post: The parent post that the method is being called on. It will be added to the StringBuilder and then this method is called on each of its children
     * @param depth - int: how many parents a post has, used to control the indenting
     * @param postChildrenDetails - StringBuilder: The current string containing details of the post and its children, will be added to in this method
     * @throws PostIDNotRecognisedException
     */
    private void recursivePost(Post post, int depth, StringBuilder postChildrenDetails) throws PostIDNotRecognisedException{
        ArrayList<Comment> childrenPosts = new ArrayList<>();
        //if depth = 0, the post is the original post, and so does not need to be altered
        if (depth != 0){
            //adds the indent for the | > that is put before each post
            for(int i =1; i<depth; i++){
                postChildrenDetails.append("\t");
            }
            //put in the | > that links a post to its reply
            postChildrenDetails.append("| >");
            //if the current post doesn't refer to a deleted post it is displayed as normal
            if (post.getPostType() != "DeletedPost") {
                //go through each line, and indent it before adding it to the StringBuilder
                Scanner scanner = new Scanner(showIndividualPost(post.getId()));
                postChildrenDetails.append("\t");
                postChildrenDetails.append(scanner.nextLine() + "\n");
                while(scanner.hasNextLine()) {
                    for(int i =0; i<depth; i++){
                        postChildrenDetails.append("\t");
                    }
                //after indenting each line based on depth, add it to the StringBuilder
                    postChildrenDetails.append(scanner.nextLine() + "\n");
            }
         //close the scanner
         scanner.close();
         }
         //if the post has been deleted, just display the dummy message given to deleted posts
         else {
            postChildrenDetails.append("\t");
            postChildrenDetails.append(post.getMesssage() + "\n");
         }
      }
      //base case, if this post's number of comments is 0, exit the recursion
      if (post.getNumberOfComments()==0){
         //check if there is a deleted comment; this may have comments under it which should stil be
             displayed
         boolean hasDeletedComment = false;
         for (Comment c: deletedComments){
            if (c.getOriginalPostID() == post.getId()){
               hasDeletedComment = true;
            }
         }
         if (!hasDeletedComment){
            return;
         }
      }
      //add the indent for the | that goes below a post
      for(int i =0; i<depth; i++){
         postChildrenDetails.append("\t");
      }
      //add the |
      postChildrenDetails.append("| \n");
      //check all comments in the system to see if they link to the current post
      for (Account a2 : accounts) {
         ArrayList<Comment> Comments = a2.getComments();
         for(Comment c : Comments) {
            if (c.getOriginalPostID() == post.getId()) {
               childrenPosts.add(c);
            }
         }
      }
      //we must check the deleted comments too
      for (Comment deletedComment : deletedComments){
         if (deletedComment.getOriginalPostID() == post.getId()){
            childrenPosts.add(deletedComment);
         }
      }

      //sort the ArrayList into ascending order of post IDs
      childrenPosts.sort((o1, o2) -> (o1.getId()-o2.getId()));
      //recursive step, call the function on all of this post's children, increasing depth by 1 so they're
          properly indented
      for (Comment child : childrenPosts){
         recursivePost(child, depth + 1, postChildrenDetails);
      }
   }

```

```java
    @Override
    public int getNumberOfAccounts() {
        //return the size of the accounts ArrayList which is the number of active accounts
        return accounts.size();
    }

    @Override
    public int getTotalOriginalPosts() {
        //NO_OF_POSTS is a recorded attribute, but it can't be used here as it doesn't account for deleted
            posts, and it also counts endorsements and comments so their IDs are also unique. Instead we loop
            through all the posts and check their type
        int totalOriginalPosts = 0;
        for (Account a: accounts){
            for (Post p: a.getPosts()){
                //if the post type is OriginalPost, add it to totalOriginalPosts
                if (p.getPostType().equals("OriginalPost")){
                    totalOriginalPosts +=1;
                }
            }
        }
        //return totalOriginalPosts once all accounts and posts have been checked
        return totalOriginalPosts;
    }

    @Override
    public int getTotalEndorsmentPosts() {
        //works similarly to getTotalOriginalPosts(). The number of endorsements isn't kept track of, and so
            they are looped though and counted
        int totalEndorsementPosts = 0;
        for (Account a: accounts){
            for (Post p: a.getPosts()){
                //this time, check if post type is EndorsementPost
                if (p.getPostType().equals("EndorsementPost")){
                    totalEndorsementPosts +=1;
                }
            }
        }
        return totalEndorsementPosts;
    }

    @Override
    public int getTotalCommentPosts() {
        //loop through each account and count their number of comments using the getComments() method
        int totalCommentPosts = 0;
        for (Account a: accounts){
            ArrayList<Comment> comments = a.getComments();
            totalCommentPosts+= comments.size();
        }
        return totalCommentPosts;
    }

    @Override
    public int getMostEndorsedPost() {
        //set mostPopularPostId to -1, which will be returned if there are no posts in the platform. Otherwise
            maxNumberOfEndorsements will be 0 or greater
```

```java
      int mostPopularPostID = -1;
      int maxNumberOfEndorsements = -1;
      //loop though all accounts, and check each post
      for( Account a : accounts){
         ArrayList<Post> posts = a.getPosts();
         for (Post p : posts){
            //if the post has more endorsements than the current maximum, update the current maximum and set mostPopularPostID to this posts ID
            if (p.getNumberOfEndorsements() > maxNumberOfEndorsements){
               maxNumberOfEndorsements = p.getNumberOfEndorsements();
               mostPopularPostID = p.getId();
            }
         }
      }
      //return the post ID with the highest number of endorsements. If two posts have the same number, the first one will be returned
      return mostPopularPostID;
   }

   @Override
   public int getMostEndorsedAccount() {
      //works very similarly to getMostEndorsedPost()
      int maxNumberOfEndorsements = -1;
      int mostPopularAccountId = -1;
      for (Account a: accounts){
         //get the number of endorsements using the getNoOfEndorsements() method
         int sumOfEndorsements = a.getNoOfEndorsements();
         //if this number is bigger than the current maximum, update the current maximum and set the account ID to mostPopularAccountID
         if (sumOfEndorsements > maxNumberOfEndorsements){
            maxNumberOfEndorsements = sumOfEndorsements;
            mostPopularAccountId = a.getId();
         }
      }
      //once all accounts have been checked, return the ID of the account with the most endorsements
      return mostPopularAccountId;
   }

   @Override
   public void erasePlatform() {
      //handle the HandleNotRecognisedException, thrown by removeAccount()
      try{
         while (!accounts.isEmpty()) {
            //remove all accounts in the platform
            Account a = accounts.get(0);
            removeAccount(a.getHandle());
         }
      } catch (HandleNotRecognisedException e){
         //as we are only using handles already retrieved from accounts, this won't be an issue. This assertion validates this
         assert(accounts.isEmpty()) : "while loop has been exited with accounts still in the platform";
      }
      //use the reset methods to set NO_OF_ACCOUNTS and NO_OF_POSTS to 0
      Account.reset();
      Post.reset();
```

```
    //go through the deletedComments ArrayList and remove their reference, so they are removed from the
        heap by the garbage collector
    while (!deletedComments.isEmpty()){
      deletedComments.remove(0);
    }
  }

  @Override
  public void savePlatform(String filename) throws IOException {
    //create an ObjectOutputStream using the filename passed in. This will throw an IOException if there
        is a problem
    ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filename));
    //add the accounts and deletedComments ArrayLists to this file
    out.writeObject(accounts);
    out.writeObject(deletedComments);
    //upcast the static values NO_OF_POSTS() and NO_OF_ACCOUNTS() to an Integer array, so that the ID's
        begin from the correct value when the platform is loaded
    Integer[] Numbers = {Post.getNO_OF_POSTS(), Account.getNO_OF_ACCOUNTS()};
    out.writeObject(Numbers);
    //close the output stream
    out.close();
  }

  @Override
  public void loadPlatform(String filename) throws IOException, ClassNotFoundException {
    //erase the current platform
    erasePlatform();
    //createa new input stream from the filename passed in
    ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename));
    //iterate over each line in the bytestream until it is empty (when the break statement is reached)
    while (true) {
      try {
        //use the general obj type to account for the 3 types of objects stored in the file
        Object obj = in.readObject();
        //if the object is an ArrayList, upcast it safley
        if (obj instanceof ArrayList) {
          ArrayList lst = (ArrayList) obj;
          //if the list is empty, move to the next object
          if (lst.isEmpty()) {
            continue;
          }
          //if the first value in this ArrayList is an account, this is the accounts Arraylist. Upcast
              the whole ArrayList and save to accounts
          if (lst.get(0) instanceof Account) {
            accounts = (ArrayList<Account>) lst;
          }
          //if the first value is a comment, it is the deletedComments ArrayList. Upcast and save to
              deletedComments
          if (lst.get(0) instanceof Comment) {
            deletedComments = (ArrayList<Comment>) lst;
          }
        }
        //otherwise, it is the list containing NO_OF_POSTS and NO_OF_ACOUNTS
        if (obj instanceof Integer[]) {
          //upacst the object to a list of Integers
```

```
               Integer[] intlst = (Integer[]) obj;
               //the 0th index is NO_OF_POSTS and the 1st is NO_OF_ACCOUNTS, save these to the platform
               Post.setNO_OF_POSTS(intlst[0]);
               Account.setNO_OF_ACOUNTS(intlst[1]);
               }
         //if there are no more objects, exit the while loop
         } catch (IOException e) {
            break;
         }
      }
      //close the input stream
      in.close();
   }
}
```