1 SocialMedia.java

```java
package socialmedia;

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

/**
 * An implementor of the SocialMediaPlatform interface
 *
 * @author 7720033503 720040807
 */
public class SocialMedia implements SocialMediaPlatform {
    ArrayList<Account> listOfAccounts = new ArrayList<Account>();
    ArrayList<Post> listOfPosts = new ArrayList<Post>();
    Post emptyPost = new Post(-1, "The original content was removed from the system and is no longer
        available.");
    // emptyPost has ID 0

    @Override
    public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
        return createAccount(handle, "");
    }

    @Override
    public int createAccount(String handle, String description) throws IllegalHandleException,
        InvalidHandleException {
        if (isValidHandle(handle)) {
            Account temporary = new Account(handle, description);
            listOfAccounts.add(temporary);
            return temporary.getID();
        }
        return 0;

    }

    /**
     * Check if a handle is valid
     *
     * @param handle - the handle to check
     * @return true if the handle is valid
     * @throws IllegalHandleException - if the handle is already in use
     * @throws InvalidHandleException - if the handle does not follow valid handle
     *                                rules
     */
    private boolean isValidHandle(String handle) throws IllegalHandleException, InvalidHandleException {
        if ((handle == null) || (handle.length() > 30) || (isWhiteSpace(handle))) {
            throw new InvalidHandleException();
        } else if (handleAlreadyExists(handle)) {
            throw new IllegalHandleException();
        } else {
            return true;
        }

    }

    /**
     * Checks if a handle contains a whitespace character
     *
     * @param handle - the handle to check
     * @return boolean - true if there is a whitespace, false if there is not
     */
    private boolean isWhiteSpace(String handle) {
        for (int i = 0; i < handle.length(); i++) {
            if (handle.charAt(i) == ' ') {
                return true;
            }
        }
        return false;
    }

    /**
     * Check if a handle already exists
     *
     * @param handle - the handle to check
     * @return a boolean - true if the handle exists, false if the handle does not
     *         exist
     */
    private boolean handleAlreadyExists(String handle) {
        try {
            findAccountIndex(handle);
            return true;
        } catch (HandleNotRecognisedException e) {
            return false;
        }
    }

    /**
     * finds the index of an account in listOfAccounts from its handle
     *
     * @param handle - the handle of the account
     * @return the index of the account
     * @throws HandleNotRecognisedException - if the handle does not currently
     *                                       belong to any account
     */
    private int findAccountIndex(String handle) throws HandleNotRecognisedException {
        // linear search
        System.out.println(handle);
        for (int i = 0; i < listOfAccounts.size(); i++) {
            System.out.println(listOfAccounts.get(i).getHandle());
            if (handle.equals(listOfAccounts.get(i).getHandle())) {
                return i;
            }
        }
        throw new HandleNotRecognisedException();
    }

    @Override
    public void removeAccount(String handle) throws HandleNotRecognisedException {
        int index = findAccountIndex(handle);
        int accountID = listOfAccounts.get(index).getID();
        listOfAccounts.remove(index);
        int counter = 0;
        while (counter < listOfPosts.size()) {
            if (listOfPosts.get(counter).getAuthorID() == accountID) {
                try {
                    deletePost(listOfPosts.get(counter).getID());
                } catch (PostIDNotRecognisedException e) {
                    assert (false);// should never happen because of how we found the IDs
                }
            } else {
                counter++;
            }
        }
    }

    @Override
    public void removeAccount(int ID) throws AccountIDNotRecognisedException {
        int index = -1;
        // find index of the account
        for (int i = 0; i < listOfAccounts.size(); i++) {
            if (listOfAccounts.get(i).getID() == ID) {
                index = i;
            }
        }
        if (index == -1) {
            throw new AccountIDNotRecognisedException();
        }
        listOfAccounts.remove(index);
        int counter = 0;
        while (counter < listOfPosts.size()) {
            if (listOfPosts.get(counter).getAuthorID() == ID) {

                try {
                    deletePost(listOfPosts.get(counter).getID());
                } catch (PostIDNotRecognisedException e) {
                    assert (false);// should never happen because of how we found the IDs
                }
            } else {
                counter++;// after because indices will change as items are removed
            }
        }
    }

    @Override
    public void updateAccountDescription(String handle, String description) throws
         HandleNotRecognisedException {
        int index = findAccountIndex(handle);
        if (index == -1) {
            throw new HandleNotRecognisedException();
        }
        listOfAccounts.get(index).setDescription(description);
        assert (listOfAccounts.get(index).getDescription() == description) : "description not set properly";
    }

    @Override
    public void changeAccountHandle(String oldHandle, String newHandle)
            throws HandleNotRecognisedException, IllegalHandleException, InvalidHandleException {
        if (isValidHandle(newHandle)) {
            int index = findAccountIndex(oldHandle);
            if (index == -1) {
                throw new HandleNotRecognisedException();
            }
            listOfAccounts.get(index).setHandle(newHandle);
        }
    }

    @Override
    public int createPost(String handle, String message) throws HandleNotRecognisedException,
         InvalidPostException {
        int index = findAccountIndex(handle);
        if (index == -1) {
            throw new HandleNotRecognisedException();
        }
        if (isValidMessage(message) == false) {
            throw new InvalidPostException();
        }
        // use a temporary post before adding it to the list
        Post tempPost = new Post(listOfAccounts.get(index).getID(), message);
        listOfPosts.add(tempPost);
        return tempPost.getID();
    }

    /**
     * Check if a message is valid within the system
     *
     * @param message - the message to check
     * @return boolean - true if the message is valid, false if the message is not
     *         valid
     */
    private boolean isValidMessage(String message) {
        if (message.length() > 100) {
            return false;
        } else if (message == "") {
            return false;
        }
        return true;
    }

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

    /**
```

```java
     * finds the index of a post in listOfPosts from its ID
     *
     * @param id - the id of the post
     * @return the index of the post
     * @throws PostIDNotRecognisedException - if no post has that ID
     */
    private int findPostIndex(int id) throws PostIDNotRecognisedException {
        int counter = 0;
        while (counter < listOfPosts.size()) {
            if (listOfPosts.get(counter).getID() == id) {
                return counter;
            }
            counter++;
        }
        throw new PostIDNotRecognisedException();
    }

    @Override
    public int endorsePost(String handle, int id)
            throws HandleNotRecognisedException, PostIDNotRecognisedException, NotActionablePostException {
        int accountIndex = findAccountIndex(handle);// throws HandleNotRecognisedException
        int postIndex = findPostIndex(id);// throws PostIDNotRecognisedException
        if (listOfPosts.get(postIndex) instanceof Endorsement) {
            throw new NotActionablePostException();
        }
        int accountID = listOfAccounts.get(accountIndex).getID();
        String endorsedAccountHandle =
             listOfAccounts.get(listOfPosts.get(postIndex).getAuthorID()).getHandle();
        // build endorsement message
        String message = "EP@" + endorsedAccountHandle + ": " + listOfPosts.get(postIndex).getMessage();
        Endorsement tempEndorsement = new Endorsement(accountID, message, id);
        listOfPosts.add(tempEndorsement);
        return tempEndorsement.getID();
    }

    @Override
    public int commentPost(String handle, int id, String message)
            throws HandleNotRecognisedException, PostIDNotRecognisedException, NotActionablePostException,
            InvalidPostException {
        int accountIndex = findAccountIndex("my_second_handle");// throws HandleNotRecognisedException
        int postIndex = findPostIndex(id);// throws PostIDNotRecognisedException
        if (listOfPosts.get(postIndex) instanceof Endorsement) {
            throw new NotActionablePostException();
        }
        if (!isValidMessage(message)) {
            throw new InvalidPostException();
        }
        Comment tempComment = new Comment(listOfAccounts.get(accountIndex).getID(), message, id);
        listOfPosts.add(tempComment);
        return tempComment.getID();
    }

    @Override
    public void deletePost(int id) throws PostIDNotRecognisedException {
        int counter = 0;
        while (counter < listOfPosts.size()) {
            // delete endorsements related to the post
            if (listOfPosts.get(counter) instanceof Endorsement) {
                Endorsement tempEndorsement = (Endorsement) listOfPosts.get(counter);
                if (tempEndorsement.getReferenceID() == id) {
                    listOfPosts.remove(counter);
                    counter--;
                }
            } // set comments to point to emptyPost
            else if (listOfPosts.get(counter) instanceof Comment) {
                Comment tempComment = (Comment) listOfPosts.get(counter);
                if (tempComment.getReferenceID() == id) {
                    tempComment.zeroReferenceID();
                }
            }
            counter++;
        }

        // delete post
        int index = findPostIndex(id);
        listOfPosts.remove(index);

    }

    /**
     * get the number of endorsements that a particular post has
     *
     * @param id - the id of the post
     * @return the number of endorsements that it has
     */
    private int getNumEndorsements(int id) {
        int counter = 0;// number of endorsements
        // iterate through the posts
        for (int i = 0; i < listOfPosts.size(); i++) {
            // check if it's an endorsement
            if (listOfPosts.get(i) instanceof Endorsement) {
                Endorsement tempEndorsement = (Endorsement) listOfPosts.get(i);
                // check if it points to the post
                if (tempEndorsement.getReferenceID() == id) {
                    counter++;
                }
            }
        }
        return counter;
    }

    /**
     * get the number of comments that a particular post has
     *
     * @param id - the id of the post
     * @return the number of comments
     */
    private int getNumComments(int id) {
        int counter = 0;// the number of comments
        // iterate through the list of posts
        for (int i = 0; i < listOfPosts.size(); i++) {
            // check if it's a comment
            if (listOfPosts.get(i) instanceof Comment) {
                Comment tempComment = (Comment) listOfPosts.get(i);
                // check if it points to the post
                if (tempComment.getReferenceID() == id) {
                    counter++;
                }
            }
        }
        return counter;
    }

    @Override
    public String showAccount(String handle) throws HandleNotRecognisedException {
        Account thisAccount = listOfAccounts.get(findAccountIndex(handle));
        // format string
        String outputString = "ID: " + Integer.toString(thisAccount.getID()) + "\n"
                + "Handle: " + handle + "\n"
                + "Description: " + thisAccount.getDescription() + "\n";
        // num posts + endorsements on posts
        int postCount = 0;
        int endorseCount = 0;
        for (int i = 0; i < listOfPosts.size(); i++) {
            if (listOfPosts.get(i).getAuthorID() == thisAccount.getID()) {
                postCount++;
                endorseCount += getNumEndorsements(listOfPosts.get(i).getID());
            }
        }
        // back to formatting string
        outputString += "Post count: " + Integer.toString(postCount) + "\n"
                + "Endorse count: " + Integer.toString(endorseCount) + "\n";
        return outputString;
    }

    @Override
    public int getTotalEndorsmentPosts() {
        int counter = 0;
        for (int i = 0; i < listOfPosts.size(); i++) {
            if (listOfPosts.get(i) instanceof Endorsement) {
                counter++;
            }
        }
        return counter;
    }

    @Override
    public int getTotalCommentPosts() {
        int counter = 0;
        for (int i = 0; i < listOfPosts.size(); i++) {
            if (listOfPosts.get(i) instanceof Comment) {
                counter++;
            }
        }
        return counter;
    }

    @Override
    public int getTotalOriginalPosts() {
        int counter = 0;
        for (int i = 0; i < listOfPosts.size(); i++) {
            // check if it is (not endorsement) and (not comment)
            if (!(listOfPosts.get(i) instanceof Endorsement) && !(listOfPosts.get(i) instanceof Comment)) {
                counter++;
            }
        }
        return counter;
    }

    @Override
    public void erasePlatform() {
        listOfAccounts.clear();
        listOfPosts.clear();
    }

    @Override
    public String showIndividualPost(int id) throws PostIDNotRecognisedException {
        // find the post
        Post thisPost;
        if (id == 0) {// the empty post
            thisPost = emptyPost;
        } else {
            thisPost = listOfPosts.get(findPostIndex(id));
        }
        return showPostFormat(id, thisPost, "");
    }

    /**
     * Format the string for showIndividualPost or showPostChildrenDetails
     *
     * @param id        - the id of the post
     * @param thisPost - the post itself (Post object)
     * @param spaces - this is here to be used when called from
     *                 showPostChildrenDetails, it is set to "" when called from
     *                 showIndividualPost
     * @return the formatted string
     */
    private String showPostFormat(int id, Post thisPost, String spaces) {
        // format string
        String outputString = "ID: " + Integer.toString(id) + "\n"
                + spaces + "Account: " + listOfAccounts.get(thisPost.getAuthorID()).getHandle() + "\n"
                + spaces + "No. endorsements: " + Integer.toString(getNumEndorsements(id))
                + spaces + " | No. comments: " + Integer.toString(getNumComments(id)) + "\n"
                + spaces + thisPost.getMessage() + "\n";
        return outputString;
    }

    @Override
    public StringBuilder showPostChildrenDetails(int id)
            throws PostIDNotRecognisedException, NotActionablePostException {// recursive
        int index = findPostIndex(id);
        if (listOfPosts.get(index) instanceof Endorsement) {// Check the post is not an endorsement
            throw new NotActionablePostException();
        }
        StringBuilder str = new StringBuilder();
        String spaces = "";
        showChildrenRecursive(str, id, spaces);
        return str;
    }

    /**
     * The recursive algorithm that forms the basis of showPostChildrenDetails
     * It has to be recursive so that spaces can be incremented depending on the
     * level of comment
     *
     * @param str   - the StringBuilder object that is being added onto
     * @param id    - the id of the post that is currently being looked at
     * @param spaces - the string of spaces that go before the post
     * @throws PostIDNotRecognisedException - if the ID isn't an existing post. this
     *                                      should never occur.
     */
    private void showChildrenRecursive(StringBuilder str, int id, String spaces)
            throws PostIDNotRecognisedException {
        str.append(showPostFormat(id, listOfPosts.get(findPostIndex(id)), spaces));
        spaces += "    ";
        // iterate through listOfPosts to find all comments pointing to the post
        for (int i = 0; i < listOfPosts.size(); i++) {
            if (listOfPosts.get(i) instanceof Comment) {
                Comment tempComment = (Comment) listOfPosts.get(i);
                if (tempComment.getReferenceID() == id) {
                    // tempComment points to the post
                    str.append(spaces + "|\n");
                    str.append(spaces + "| > ");
                    // call this function with tempComment
                    showChildrenRecursive(str, tempComment.getID(), spaces);
                }
            }
        }
    }

    @Override
    public int getMostEndorsedPost() {
        int maxEndorsement = 0;
        int maxEndorsedID = 0;
        int endorsementCount;
        int thisPostID;
        // iterate through listOfPosts in order to find the maximum number of
        // endorsements
        for (int i = 0; i < listOfPosts.size(); i++) {
            thisPostID = listOfPosts.get(i).getID();
            endorsementCount = getNumEndorsements(thisPostID);
            if (maxEndorsement < endorsementCount) {
                maxEndorsedID = thisPostID;
                maxEndorsement = endorsementCount;
            }
```

```java
        }
        return maxEndorsedID;
    }

    @Override
    public int getMostEndorsedAccount() {
        int maxEndorsement = 0;
        int maxEndorsedID = 0;
        int endorsementCount;
        int thisAccountID;
        // iterate through listOfAccounts in order to find the maximum number of
        // endorsements
        for (int i = 0; i < listOfAccounts.size(); i++) {
            endorsementCount = 0;
            thisAccountID = listOfAccounts.get(i).getID();
            for (int j = 0; j < listOfPosts.size(); j++) {
                if (listOfPosts.get(j).getAuthorID() == thisAccountID) {
                    endorsementCount++;
                    endorsementCount += getNumEndorsements(listOfPosts.get(j).getID());
                }
                if (endorsementCount > maxEndorsement) {
                    maxEndorsedID = thisAccountID;
                    maxEndorsement = endorsementCount;
                }
            }
        }
        return maxEndorsedID;
    }

    @Override
    public void savePlatform(String filename) throws IOException {
        ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filename));
        out.writeObject(listOfPosts);
        out.writeObject(listOfAccounts);
        out.close();
    }

    @Override
    public void loadPlatform(String filename) throws IOException, ClassNotFoundException {
        erasePlatform();
        ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename));
        listOfPosts = (ArrayList) in.readObject();
        listOfAccounts = (ArrayList) in.readObject();
        // set the counter for the IDs for Post and Account from the ID of the latest in
        // the list
        Post.setIDOnLoad(listOfPosts.get(listOfPosts.size() - 1).getID());
        Account.setIDOnLoad(listOfAccounts.get(listOfAccounts.size() - 1).getID());
        in.close();
    }
}
```

2 Account.java

```java
package socialmedia;
```

```java

import java.io.Serializable;

/**
 * Account class
 * Attributes:
 * static counter - used to count the IDs to ensure they are unique
 * ID - unique int for each Account (starts from 0)
 * handle - unique account handle
 * desc - account description
 *
 * @author - 7720033503 720040807
 */

public class Account implements Serializable {
    protected static int counter = 0;
    private int ID;// unique
    // ID starts from 0 for Account class
    private String handle;// unique also
    private String description;

    /**
     * Account creation method
     *
     * @param handle - the unique handle of the account
     * @param desc - the description of the account, may be an empty string
     */
    public Account(String handle, String desc) {
        this.ID = counter;
        counter++;
        this.handle = handle;
        this.description = desc;
    }

    // setters and getters ******************************

    /**
     * set counter (which the IDs count from), used only when loading
     *
     * @param maxID - ID of the last created account. next ID will be maxID+1
     */
    public static void setIDOnLoad(int maxID) {
        counter = maxID + 1;
    }

    /**
     * getter for ID
     *
     * @return ID
     */
    public int getID() {
        return ID;
    }

    /**
```

```java
     * getter for handle
     *
     * @return handle
     */
    public String getHandle() {
        return handle;
    }

    /**
     * setter for handle
     *
     * @param newHandle - must be a valid handle
     */
    public void setHandle(String newHandle) {// not validated here
        handle = newHandle;
    }

    /**
     * getter for description
     *
     * @return description
     */
    public String getDescription() {
        return description;
    }

    /**
     * setter for description
     *
     * @param newDescription
     */
    public void setDescription(String newDescription) {
        description = newDescription;
    }
}
```

3 Post.java

```java
package socialmedia;

import java.io.Serializable;

/**
 * Post class
 * Attributes:
 * static counter - used to count the IDs to ensure they are unique
 * ID - unique int for each Post (starts from 0)
 * message - must be <=100chars
 * authorID - ID of the account that the post is associated with
 *
 * @author - 7720033503 720040807
 */

public class Post implements Serializable {
    protected static int counter = 0;
    protected int ID;// unique
    // IDs for normal posts start at 1 - 0 is emptyPost
    protected String message;// <=100chars
    protected int authorID;

    public Post(int authorID, String message) {// valid message - validated in createPost
        this.ID = counter;
        counter++;
        this.authorID = authorID;
        this.message = message;
    }

    // setters and getters ******************************

    /**
     * set counter (which the IDs count from), used only when loading
     *
     * @param maxID - ID of the last created account. next ID will be maxID+1
     */
    public static void setIDOnLoad(int maxID) {
        counter = maxID + 1;
    }

    /**
     * getter for ID
     *
     * @return ID
     */
    public int getID() {
        return ID;
    }

    /**
     * getter for authorID
     *
     * @return authorID
     */
    public int getAuthorID() {
        return authorID;
    }

    /**
     * getter for message
     *
     * @return message
     */
    public String getMessage() {
        return message;
    }

}
```

4 Comment.java

```java
package socialmedia;

/**
 * Comment inherits from Post
 * Has additional attribute:
 * referenceID - the ID of the post that the comment is referencing
 */
public class Comment extends Post {
    protected int referenceID;// post that the comment is referencing

    /**
     * Constructor for Comment
     *
     * @param accountID - the account making the comment
     * @param message   - the message that the comment will carry
     * @param referenceID - the ID of the post that the comment is referencing
     */
    public Comment(int accountID, String message, int referenceID) {
        super(accountID, message);
        this.referenceID = referenceID;
    }

    /**
     * getter for referenceID
     *
     * @return referenceID
     */
    public int getReferenceID() {
        return referenceID;
    }

    /**
     * set the referenceID to 0
     * this points it to an empty post
     * to be called in deletePost
     */
    public void zeroReferenceID() {
        referenceID = 0;
    }
}
```

5 Endorsement.java

```java
package socialmedia;

/**
 * Endorsement inherits from Post
 * Has additional attribute:
 * referenceID - the ID of the post that the endorsement is referencing
 */
public class Endorsement extends Post {
    protected int referenceID;// post that the endorsement is referencing

    /**
     * Constructor for Endorsement
     *
     * @param accountID - the account making the endorsement
     * @param message   - the message that the endorsement will carry
     * @param referenceID - the ID of the post that the endorsement is referencing
     */
    public Endorsement(int accountID, String message, int referenceID) {
        super(accountID, message);
        this.referenceID = referenceID;
    }

    /**
     * getter for referenceID
     *
     * @return referenceID
     */
    public int getReferenceID() {
        return referenceID;
    }
}
```