```java
/**
 * This is an implementation of the BankProgram interface that uses
 * an array to store the data.
 *--------------------------------
 * CSE222_HW01_101044044
  *--------------------------------
 * @author Samet Sait Talayhan
 */
package bankprogram;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;


public class ArrayBasedBP implements BankProgram{
    // Data Fields

    /**
     * The initial capacity of the array
     */
    private static final int INITIAL_CAPACITY = 100;
    /**
     * The current capacity of the array
     */
    private int customersCapacity = INITIAL_CAPACITY;
    /**
     * The current size of the array (number of customer entries)
     */
    private int customersSize = 0;

    private int bankWorkersCapacity = INITIAL_CAPACITY;
    /**
     * The current size of the array (number of bank worker entries)
     */
    private int bankWorkersSize = 0;

    /**
     * The array to contain the customers data
     */
    private Customer[] theCustomers =
            new Customer[customersCapacity];

    /**
     * The array to contain the bank worker data
     */
    private BankWorker[] theBankWorkers =
            new BankWorker[customersCapacity];

    /**
     * The data file that contains the customers data
     */
    private String customersTxt = null;

    /**
     * The data file that contains the bank workers data
     */
    private String bankWorkersTxt = null;

    /**
     * Boolean flag to indicate whether the program was modified since it was
     * either loaded or saved.
     */
    private boolean modifiedAccount = false;
    /**
     * Boolean flag to indicate whether the program was modified since it was
     * either loaded or saved.
     */
    private boolean modifiedWorker = false;

    /**
     * Method to load the data file. pre: The program storage has been created
     * and it is empty. If the file exists, it consists of name-account number-
     * account balance on adjacent lines.
     * post: The data from the file is loaded into the program.
     *
     * @param customersTxt The name of the data file for customers.
     * @param bankWorkersTxt The name of the data file for bank workers.
     */
    @Override
    public void loadData(String customersTxt, String bankWorkersTxt) {
        // Remember the source name.
        this.customersTxt = customersTxt;
        this.bankWorkersTxt = bankWorkersTxt;
        try {
            // Create a BufferedReader for the file.
            BufferedReader in = new BufferedReader(
                    new FileReader(customersTxt));
            String name;
            String accountNumber;
            String accountBalance;

            // Read each name and number and add the entry to the array.
            while ((name = in.readLine()) != null) {
                // Read name,accountNumber and accountBalance
                // from successive lines.
                if ((accountNumber = in.readLine()) == null) {
                    break; // No accountNumber read, exit loop.
                }
                if ((accountBalance = in.readLine()) == null) {
                    break; // No accountBalance read, exit loop.
                }
                // Add an entry for this name, accountNumber and
                // accountBalance.
                add(name, accountNumber, accountBalance);
            }

            // Close the file.
            in.close();

            in = new BufferedReader(new FileReader(bankWorkersTxt));

            // Read each name and number and add the entry to the array.
            while ((name = in.readLine()) != null) {
                // Add an entry for this name and number.
                addWorker(name);
            }

            // Close the file.
            in.close();
        } catch (FileNotFoundException ex) {
        } catch (IOException ex) {
            System.err.println("Load of directory failed.");
            System.exit(1);
        }
    }

    /**
     * Add an entry or change an existing entry.
     *
     * @param name The name of the customer being added or changed
     * @param accountNumber The new account number to be assigned
     * @param accountBalance The new account number to be assigned
     * @return The old accountNumber or, if a new entry, null
     */
    @Override
    public String addOrChangeAccount(String name, String accountNumber,
            String accountBalance)
    {
        String oldAccountNumber = null;
        int index = find(accountNumber);
        if (index > -1) {
            oldAccountNumber = theCustomers[index].getAccountNumber();
            theCustomers[index].setAccountNumber(accountNumber);
        } else {
            add(name, accountNumber, accountBalance);
        }
        modifiedAccount = true;
        return oldAccountNumber;
    }

     /**
     * Add an entry or change an existing entry.
     *
     * @param name The name of the worker being added or changed
     * @return The old worker name or, if a new entry, null
     */
    @Override
    public String addOrChangeWorker(String workerName)
    {
        String oldWorkerName = null;
        int index = findWorker(workerName);
        if (index > -1) {
            oldWorkerName = theBankWorkers[index].getName();
            theBankWorkers[index].setName(workerName);
        } else {
            addWorker(workerName);
        }
        modifiedWorker = true;
        return oldWorkerName;
    }

    /**
     * Look up an account.
     *
     * @param accountNumber
     * @return The account balance. If not in the program, null is returned
     */
    @Override
    public String lookupAccount(String accountNumber)
    {
        int index = find(accountNumber);
        if (index > -1) {
            return theCustomers[index].getAccountBalance();
        } else {
            return null;
        }
    }

     /**
     * Look up a bank worker.
     *
     * @param accountNumber
     * @return The bank worker, If not in the program, null is returned
     */
    @Override
    public String lookupBankWorker(String workerName)
    {
        int index = findWorker(workerName);
        if (index > -1) {
            return theBankWorkers[index].getName();
        } else {
            return null;
        }
    }

    /**
     * Method to save the program. pre: The program has been loaded with
     * data. post: Contents of array written back to the file in the form of
     * name-account number-account balance on adjacent lines.
     * modified is reset to false.
     */
    @Override
    public void save()
    {
        if (modifiedAccount)
        { // If not modified, do nothing.
            try
            {
                try (PrintWriter out = new PrintWriter(
                             new FileWriter(customersTxt))) {
                    for (int i = 0; i < customersSize; i++) {
                        // Write the name.
                        out.println(theCustomers[i].getName());
                        // Write the account number.
                        out.println(theCustomers[i].getAccountNumber());
                        // Write the account balance.
                        out.println(theCustomers[i].getAccountBalance());
                    }
                }
                modifiedAccount = false;
            } catch (Exception ex) {
                System.err.println("Save of accounts file failed");
                System.exit(1);
            }
        }
        if(modifiedWorker)
        {// If not modified, do nothing.
            try
            {
                try (PrintWriter out = new PrintWriter(
                             new FileWriter(bankWorkersTxt))) {
                    for (int i = 0; i < bankWorkersSize; i++) {
                        // Write the name.
                        out.println(theBankWorkers[i].getName());
                    }
                }
                modifiedWorker = false;
            } catch (Exception ex) {
                System.err.println("Save of workers file failed");
                System.exit(1);
            }
        }//end of if
    }

    /**
     * Find an account in the data base.
     *
     * @param accountNumber The number to be found
     * @return The index of the entry with the requested number. If the
     * accountNumber is not in the data base, returns -1
     */
    private int find(String accountNumber)
    {
```

```java
        for (int i = 0; i < customersSize; i++)
        {
            if (theCustomers[i].getAccountNumber().equals(accountNumber))
            {
                return i;
            }
        }
        return -1; // Account number not found.
    }

     /**
     * Find a bank worker in the data base.
     *
     * @param workerName The name to be found
     * @return The index of the entry with the requested name. If the
     * workerName is not in the data base, returns -1
     */
    private int findWorker(String workerName)
    {
        for (int i = 0; i < bankWorkersSize; i++)
        {
            if (theBankWorkers[i].getName().equals(workerName))
            {
                return i;
            }
        }
        return -1; // Account number not found.
    }

    /**
     * Add a customer to the program.
     *
     * @param name The name of the new person
     * @param accountNumber The number of the new person account
     * @param accountBalance The balance of the new person account
     */
    private void add(String name, String accountNumber,String accountBalance)
    {
        if (customersSize >= customersCapacity) {
            reallocate();
        }
        theCustomers[customersSize] = new Customer(name, accountNumber, accountBalance);
        customersSize++;
    }

     /**
     * Add a worker to the program.
     *
     * @param bankWorkerName The name of the new bank worker
     */
    private void addWorker(String bankWorkerName)
    {
        if (bankWorkersSize >= bankWorkersCapacity) {
            reallocateWorker();
        }
        theBankWorkers[bankWorkersSize] = new BankWorker(bankWorkerName);
        bankWorkersSize++;
    }

     /**
     * Allocate a new array to hold the bank workers array.
     */
    private void reallocateWorker() {
        bankWorkersCapacity *= 2;
        BankWorker[] newBankWorkers = new BankWorker[bankWorkersCapacity];
        System.arraycopy(theBankWorkers, 0, newBankWorkers, 0,
                theBankWorkers.length);
        theBankWorkers = newBankWorkers;
    }

    /**
     * Allocate a new array to hold the customers array.
     */
    private void reallocate() {
        customersCapacity *= 2;
        Customer[] newCustomers = new Customer[customersCapacity];
        System.arraycopy(theCustomers, 0, newCustomers, 0,
                theCustomers.length);
        theCustomers = newCustomers;
    }

    /**
     * Remove a account from the program.
     *
     * @param accountNumber - The number of the account to be removed.
     * @return The current account number. If not in file, null is returned.
     */
    @Override
    public String removeAccount(String accountNumber) {
        int index = find(accountNumber);
        // Wrong. if (index < size) {
        if (index > -1) {
            String returnValue = theCustomers[index].getAccountNumber();
            remove(index);
            modifiedAccount = true;
            return returnValue;
        } else {
            return null;
        }
    }

    /**
     * Remove a account from the program.
     *
     * @param index The index of the item to be removed
     */
    private void remove(int index) {
        for (int i = index; i < customersSize - 1; i++) {
            theCustomers[i] = theCustomers[i + 1];
        }
        --customersSize;
    }
}
```