```
package data;

import model.User;
import model.Car;
import model.Cash;
import model.CreditCard;
import model.MobilePay;
import model.Booking;
import model.Payment;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * This class handles read and write activities related to the external CSV
    files.
 *
 * Setting a custom delimiter (;), CSV files are easier to manipulate via
    external tools
 * (e.g. MS Excel) and allow for white space in the values.
 *
 */
public class Database {
    private static List<User> users             = new ArrayList<User>();
    private static List<Car> cars           = new ArrayList<Car>();
    private static List<Booking> bookings   = new ArrayList<Booking>();
    private static List<Payment> payments   = new ArrayList<Payment>();

    public static List<User> getUsers() {
        return users;
    }

    public static List<Car> getCars() {
        return cars;
    }

    public static List<Booking> getBookings() {
        return bookings;
    }

    public static List<Payment> getPayments() {
        return payments;
    }

    public static void addUser(User newUser) {
        users.add(newUser);
    }

    public static void addCar(Car newCar) {
        cars.add(newCar);
    }
```

```
public static void addBooking(Booking newBooking) {
    bookings.add(newBooking);
}

public static void addPayment(Payment newPayment) {
    payments.add(newPayment);
}


/**
 * Write a line to a file
 *
 * @param table The name of the table / file to write to
 * @param line The content of the new line
 */
public static Boolean write(String table, String line) {
    try {
        BufferedWriter writer = new BufferedWriter(new
            OutputStreamWriter(new FileOutputStream("src/
            data/"+table+".csv", true), "utf-8"));
        PrintWriter out = new PrintWriter(writer);
        out.println(line);
        out.close();
        return true;
    } catch (IOException ex) {
        return false;
    }
}

/**
 * Truncate (= clear) table method
 *
 * @param table Name of the table to be truncated
 */
public static void truncate(String table) {
    if ( table == "bookings" ) {
        bookings = new ArrayList<Booking>();
    } else if ( table == "cars" ) {
        cars = new ArrayList<Car>();
    } else if ( table == "users" ) {
        users = new ArrayList<User>();
    } else if ( table == "payment" ) {
        payments = new ArrayList<Payment>();
    }
}

/**
 * Delete a object from a table by the object's ID
 *
 * @param table The table which will be deleted from
 * @param searchId The ID of the object that will be deleted
 */
// return boolean, print in UI
public static Boolean deleteById(String table, int searchId) {
    return Database.manipulate(table, searchId, null);
}

/**
 * Edit a object in a table by the object's ID
 *
```

```
     * @param table The table which will be edited
     * @param searchId The ID of the object that will be edited
     * @param newLine
     */
    // see above
    public static Boolean editById(String table, int searchId, String
        newLine) {
        return Database.manipulate(table, searchId, newLine);
    }

    /**
     * Method to manipulate (delete and edit) a table.
     * Copies parts of the existing table to a new temporary file, then
         deletes the
     * old file and renames the new one.
     *
     * Critical problem with this approach: When multiple manipulations are
         executed
     * within a short period of time, this might lead to data loss because
         the reading
     * and writing may take longer than the copying and renaming.
     *
     * NOT PRODUCTION READY!
     *
     * @param table The name of the table that will be manipulated
     * @param searchId The ID of the object that will be manipulated
     * @param newLine The content of the new line
     * @return A boolean variable indication success (true) or failure
         (false)
     */
    public static Boolean manipulate(String table, int searchId, String
        newLine) {
        boolean success = false;

        try {
            String fileName     = "src/data/"+table+".csv";
            File inputFile  = new File(fileName);
            File tempFile   = new File(table+"Temp.csv");

            BufferedWriter writer = new BufferedWriter(new FileWriter
                (tempFile));

            // Read File, set delimiter to read CSV correctly
            @SuppressWarnings("resource")
            Scanner readFile    = new Scanner(inputFile).useDelimiter("\\s*;
                \\s*");
            // Get header of that table
            String header       = readFile.nextLine();
            // Write that header to the new file
            writer.write(header + System.getProperty("line.separator"));

            while ( readFile.hasNext() ) {
                int id      = Integer.parseInt(readFile.next());
                String line     = id + readFile.nextLine();

                if ( searchId != id ) {
                    writer.write(line + System.getProperty("line.separator")
                        );
                } else if ( newLine != null && !newLine.isEmpty() ) {
                    writer.write(newLine +
```

```
                    System.getProperty("line.separator"));
            }
        }

        writer.close();
        inputFile.delete();
        success = tempFile.renameTo(new File(fileName));
        Database.truncate(table);
        Database.read(table);
    } catch (IOException ex) {
        success = false;
    }

    return success;
}

/**
 * Read a table and create respective instances.
 *
 * This approach is a little "hacky". The last item of a line has to be
     cropped to be
 * read correctly.
 *
 * @param table Name of the table which will be read
 */
public static Boolean read(String table) {
    Boolean success = false;

    try {
        File file = new File("src/data/"+table+".csv");
        @SuppressWarnings("resource")
        Scanner readFile = new Scanner(file).useDelimiter("\\s*;\\s*");
        readFile.nextLine();

        while ( readFile.hasNext() ) {
            if ( table.equals("users") ) {
                int id              = Integer.parseInt(readFile.next());
                String role            = readFile.next();
                String name            = readFile.next();
                String streetname   = readFile.next();
                String housenumber  = readFile.next();
                int postcode        = Integer.parseInt(readFile.next());
                String dob          = readFile.next();
                String telephone    = readFile.next();
                String cpr          = readFile.next();
                String username         = readFile.next();
                String password         = readFile.nextLine();
                password            = password.substring(1, password.
                    length()-1);


                /*
                 * Debug
                 */
                //System.out.println(id+" "+name+" "+streetname+"
                    "+housenumber+" "+postcode+" "+dob+" "+telephone+"
                    "+cpr+" "+username+" "+password+".");

                User newUser = new User(id, role, name, streetname,
                    housenumber, postcode, dob, telephone, cpr, username
```

```
            , password);
        users.add(newUser);
    }
    else if ( table.equals("cars") ) {
        int id              = Integer.parseInt(readFile.next());
        int owner           = Integer.parseInt(readFile.next());
        String name             = readFile.next();
        String type             = readFile.next();
        String brand        = readFile.next();
        String transmission     = readFile.next();
        int seats           = Integer.parseInt(readFile.next());
        double rate             = Double.parseDouble(readFile.
            next());
        String description  = readFile.next();
        String location         = readFile.nextLine();
        location            = location.substring(1, location.
            length()-1);

        /*
         * Debug
         */
        //System.out.println(id+" "+name+" "+type+" "+brand+"
            "+transmission+" "+seats+" "+rate+" "+description);

        // Make users
        Car newCar = new Car(id, owner, name, type, brand,
            transmission, seats, rate, description, location);
        cars.add(newCar);
    }
    else if ( table.equals("bookings") ) {
        int id                      = Integer.parseInt(readFile.
            next());
        String date                     = readFile.next();
        int carId                   = Integer.parseInt(readFile.
            next());
        int price                   = Integer.parseInt(readFile.
            next());
        int userId                  = Integer.parseInt(readFile.
            next());
        String rentalDate           = readFile.next();
        int rentalStartTimestamp    = Integer.parseInt(readFile.
            next());
        int rentalEndTimestamp      = Integer.parseInt(readFile.
            next());
        String bookingRef           = readFile.nextLine();
        bookingRef                  = bookingRef.substring(1,
            bookingRef.length()-1);

        /*
         * Debug
         */
        //System.out.println(id+" "+name+" "+type+" "+brand+"
            "+transmission+" "+seats+" "+rate+" "+description);

        // Make users
        Booking newBooking = new Booking(id, date, carId, price,
            userId, rentalDate, rentalStartTimestamp,
            rentalEndTimestamp, bookingRef);
        bookings.add(newBooking);
    }
```

```
                else if ( table.equals("payments") ) {
                    int id                      = Integer.parseInt(readFile.
                        next());
                    String bookingRef           = readFile.next();
                    int amount                  = Integer.parseInt(readFile.
                        next());
                    String method               = readFile.next();

                    // Check for payment method
                    // Item must always contain 4 + 3 = 7 items
                    if ( method.equals("credit card") ) {
                        String number           = readFile.next();
                        String expiryDate       = readFile.next();
                        String securityCode     = readFile.nextLine();
                        securityCode = securityCode.substring(0,
                            securityCode.length()-1);

                        Payment newCreditCard = new CreditCard(id,
                            bookingRef, amount, number, expiryDate,
                            securityCode);
                        payments.add(newCreditCard);
                    } else if ( method.equals("cash") ) {
                        // Cannibalize remaining items
                        readFile.next();
                        readFile.next();
                        readFile.nextLine();

                        Payment newCash = new Cash(id, bookingRef, amount);
                        payments.add(newCash);
                    } else if ( method.equals("mobile pay") ) {
                        String mobilePhone = readFile.next();
                        // Cannibalize remaining items
                        readFile.next();
                        readFile.nextLine();

                        Payment newMobilePay = new MobilePay(id, bookingRef,
                            amount, mobilePhone);
                        payments.add(newMobilePay);
                    }
                }
            }

            success = true;
        } catch (IOException ex) {
            return success = false;
        }

        return success;
    }
}
```