Project #1 – 100 points

Due Date

Sunday, February 13, 11:59pm.

Submission

- Submit **a single zipped file** to Canvas. The zipped file MUST include the following grading items.
  (1) Source folder **src** [60 points]
    (a) Create a package to organize the source files *.java; use all lowercase letters for the package name
    (b) MUST include all team members' names using the **@author** tag in the comment block on top of EVERY Java class, or you will **lose 2 points**.
  (2) **Test design document**. You must use a document editor and include all test cases you used to test the **isValid()** in the Date class, and the **compareTo()** method in the Timeslot class. [15 points]
  (3) **Testbed main()** in the following classes implementing the test cases in the test design document in (2).
    (a) Date class [10 points]
    (b) Timeslot class [10 points]
  (4) **Javadoc** folder **doc**, including all the files generated. [5 points]
- The submission button on Canvas will disappear after **February 13, 11:59pm**. It is your responsibility to ensure your Internet connection is good for the submission. **You get 0 points** if you do not have a submission on Canvas. DO NOT wait until the last minute. DO NOT send the project to me or the graders through the emails.

Project Description

A clinic is providing the vaccination service at 5 different locations in New Jersey.

Bridgewater, 08807, Somerset County
Piscataway, 08854, Middlesex County
Princeton, 08542, Mercer County
Morristown, 07960, Morris County
Union, 07083, Union County

Your team will develop a simple software to help the clinic processes the transactions for vaccination appointments. This project uses the IDE console as the user interface and utilizes the Java standard input and output to read the transactions and write the results. A transaction in this project is defined as a line of data entered through the console. It represents an operation the user performs to manage the appointments.

A transaction is a command line that always begins with a command, in uppercase letters, and followed by several data tokens delimited by spaces. The user shall be able to book or cancel appointments. The software shall provide the functionality of displaying all the scheduled appointments sorted by the zip codes or by the patients, in ascending order. **Commands are case-sensitive**, which means the commands with lowercase letters are invalid. You are required to deal with bad commands not supported, or you will **lose 2 points** for each bad command not handled. Your software must support the following commands.

- **B** command, to **book an appointment** and add the appointment to the schedule. A schedule contains a list of appointments. Each appointment shall include the information of the patient, date, time, and location in county name. Below is a sample transaction for booking an appointment. You can assume that the user will always enter enough data tokens to book an appointment in the format shown below.

```
B 8/31/1978 Jane Doe 7/19/2022 9:15 middlesex
```

The above transaction starts with the B command, including the patient's date of birth, first name, last name, appointment date, time and the location. The dates shall be given in **mm/dd/yyyy** format, and the times shall be given in hh:mm format, where hh represent the hours in a 24-hour format, and mm represent the minutes. Appointment times are given with a 15-minute interval, the first appointment of a day is 9:00 and the last appointment of a day is 16:45. **County names are not case-sensitive**. The system shall allow a patient to schedule multiple appointments, as long as they are on different days. You should not allow the appointment to be added if one of the conditions below happens. See sample output for the error messages to display.

1. A date is not a valid calendar date.
2. The date of birth is today or a future date.
3. The appointment date is today or a date before today, or a date beyond this year.
4. The time is not a 15-minute interval and outside of the range of the appointment times of the day.
5. An appointment with the same patient, timeslot and location is already in the schedule.
6. The specified timeslot (same date and time) at the specified location has already been taken.
7. The location with the county name is not a valid location.
8. The user is booking an appointment with the same patient and date but a different location with an existing appointment.

- **C** command, to **cancel an appointment** and remove the specified appointment from the schedule, for example,

  ```
  C 8/31/1978 Jane Doe 7/19/2022 9:15 middlesex
  ```

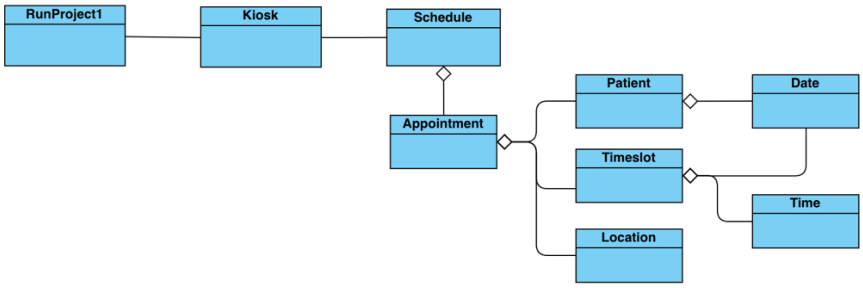
  The above transaction removes an appointment from the schedule. See sample output for the messages to display.
- **CP** command, to **cancel all appointments of a given patient** and remove all the appointments for this patient from the schedule. A patient is uniquely identified by his/her date of birth, first and last names, for example,

  ```
  CP 8/31/1978 Jane Doe
  ```
- **P** command, to **display all appointments** in the schedule to console, with the current order in the array**.**
- **PZ** command to **display all appointments** in the schedule ordered by the zip codes. If two appointments have the same zip code, display the appointment with an earlier timeslot first.
- **PP** command, to **display all appointments** in the schedule ordered by the patients. Sort by patients' last names, then first names, then dates of birth. For a patient who has multiple appointments, display the appointments based on their current order in the array; that is, the order of the appointments for the same patient doesn't matter.
- **Q** command, to stop the program execution and display "`Kiosk session ended.`"

Project Requirement

1. You MUST follow the Software Projects Coding Standard and Ground Rules posted on Canvas under "Modules" "Week #1". You will lose points if you are not following the rules.
2. There are sample input and output at the end of this document for your reference. The graders will be using the sample input as the test cases to test your project. Your project should be able to take the sample input in batch with the same sequence without getting any exceptions and without terminating abnormally. You will **lose 2 points** for each incorrect output or each exception.
3. Each Java class must go in a separate file. **-2 points** if you put more than one Java class into a file.
4. Your program MUST handle bad commands! **-2 points** for each bad command not handled.
5. You are not allowed to use any Java library classes, except **Scanner**, **StringTokenizer, Calendar** and **DecimalForamt** classes. **You will lose 5 points** for each additional Java library class imported, with a **maximum of losing 10 points**.

6. When you import Java library classes, be specific and DO NOT import unnecessary classes or import the whole package. For example, `import java.util.*;`, this will import all classes in the `java.util` package. You will **lose 2 points** for using the asterisk "*" to include all the Java classes in the `java.util` package, or other java packages, with a **maximum of losing 4 points.**
7. You MUST include the Java classes in the class diagram below. That is, at least 9 Java classes. **-5 points** for each class missing or NOT used. You CAN add necessary constructors, private methods (helper methods), and other public methods to each class. You should define necessary constant identifiers and do not use MAGIC NUMBERs. You can also create an additional class to define all the constant identifiers needed in this project.



(a) **Appointment class**

```
public class Appointment {
    private Patient  patient;
    private Timeslot slot;
    private Location location;
       ...
    @Override
    public boolean equals(Object obj) { }
       ...
    @Override
    public String toString() { }
}
```

- You CANNOT change or add instance variables for this class. **-2 points** for each violation.
- You CANNOT read from console or use **System.out** statements in this class. **-2 points** for each violation.
- **toString()** method returns a textual representation of an appointment in the following format.

```
Jane Doe, DOB: 1/19/2000, Appointment detail: 12/1/2022, 9:45, Union 07083, UNION
```

- **equals()** method returns true if the patient, timeslot and location are equal for the two appointments.
- You CANNOT change the signatures of the **toString()** and **equals()** methods. You cannot remove the **@Override** tags. **-2 points** for each violation.

(b) **Schedule class**

```
public class Schedule {
    private Appointment [] appointments;
    private int numAppts;
       ...
    private int find(Appointment appt) {} //return the index, or NOT_FOUND
    private void grow() {} //increase the capacity of the container by 4
    public boolean add(Appointment appt) {}
    public boolean remove(Appointment appt) {}
    public void print() {} //print all the appointments in current order
    public void printByZip() {} //sort by zip codes and print
    public void printByPatient() {} //sort by patient and print }
```

- This is the container class that implement an array-based linear data structure to hold the list of vaccination appointments. A new appointment is always added to the end of the list. An instance of the Schedule class is a growable container with an initial capacity of 4, and automatically grows (increases) the capacity by 4 whenever it is full. The container does not decrease in capacity. You are **NOT allowed** to use the Java library class **ArrayList** anywhere in the project, or **you will get 0 points for this project**.
- This class shall provide the methods for managing the schedule. You CANNOT change or add instance variables. **-2 points** for each violation.
- You must implement the methods listed above, and you CANNOT change the signatures of the methods. **-2 points** for each method not implemented or not used.
- You CANNOT use **System.out** in this class, EXCEPT the three **print()** methods, **-2 points** for each violation.
- The **find()** method searches an appointment in the list and returns the index if it is found, it **returns -1** if it is not in the list. You must define a constant identifier "NOT_FOUND" for the value -1.
- The **remove()** method delete an appointment from the schedule. This method maintains the relative order of the appointments in the list after the deletion, **-3 points** if this is not done correctly.
- You must use an "in-place" sorting algorithm to implement the sorting by writing the code yourself. You CANNOT use **Arrays.sort()** or **System.arraycopy()** or any Java library classes for sorting. You will **lose 10 points** if you do.
- You can add additional methods. However, the methods you added must either take no parameter or has a single parameter, which takes an instance of Appointment class, such as (`Appointment appt`), or you will **lose 2 points** for each violation.

(c) **Kiosk class**

- This class is the user interface class to process the transactions entered through the console and output the results to the console. An instance of Kiosk class can process a single transaction, as well as a batch of transactions. If it cannot process batch transactions, **you will lose 5 points**. This is an interactive program such that, it displays the results to the console whenever one or more transactions have been entered.
- When your project starts running, it shall display "`Kiosk running. Ready to process transactions.`". Next, it will read and process transactions continuously until the "Q" command is read. Before the project stops running, display "`Kiosk session ended.`".
- You must define a `public void run()` method that includes a while loop to continuously read the command lines (transactions) from the console. You will **lose 5 points** if the run() method is missing. You MUST keep this method **under 35 lines** for readability, or you will **lose 3 points**. You can define necessary instance variables and private methods for handling the commands.

(d) **RunProject1 class** is a driver class to run your Project 1. The main method will call the **run()** method in the Kiosk class.

```
public class RunProject1 {
    public static void main(String[] args) {
        new Kiosk().run();
    }
}
```

(e) **Date class**

```
public class Date implements Comparable<Date> {
    private int year;
    private int month;
    private int day;

    public Date(String date) {} //take "mm/dd/yyyy" and create a Date object
    public Date() {} //create an object with today's date (see Calendar class)
       ...
     public boolean isValid() {}
       ...
    @Override
     public int compareTo(Date date) {}
}
```

- You must implement the constructors and methods listed above. You must implement the Comparable Interface and implement the **compareTo()** method, or **lose 2 points** for each violation. You will need this method to sort by the dates.
- You CANNOT change or add instance variables, and you CANNOT use **System.out** statements in this class, EXCEPT the testbed main(), **-2 points** for each violation.
- The **isValid()** method checks if a date is a valid calendar date.
  - For the month, January, March, May, July, August, October and December, each has 31 days; April, June, September and November, each has 30 days; February has 28 days in a non-leap year, and 29 days in a leap year. DO NOT user **magic numbers** for the months, days and years. Below are some examples for defining the identifiers for the constants.

    ```
    public static final int QUADRENNIAL = 4;
    public static final int CENTENNIAL = 100;
    public static final int QUATERCENTENNIAL = 400;
    ```

  - To determine whether a year is a leap year, follow these steps:

    Step 1. If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.
    Step 2. If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.
    Step 3. If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.
    Step 4. The year is a leap year.
    Step 5. The year is not a leap year.

  - You MUST design the test cases to thoroughly test the **isValid()** method. You must write a testbed main and implement the test cases. You must follow the instructions in the "Test Design" section in the "Software Project Coding Standard and Ground Rules. In the testbed main, you MUST write code to print out the test results to the console showing the test cases are passed or failed. The testbed main for this class **is worth 10 points**. Please use "//" comments to identify the test case numbers in the testbed main.

(f) **Time class**

```
public class Time implements Comparable<Time> {
    private int hour;
    private int minute;

    public boolean isValid() { }
    @Override
    public String toString() { }
    @Override
    public int compareTo(Time time) { } }
```

- You must implement the methods listed above. You must implement the Comparable Interface and implement the **compareTo()** method, or **lose 2 points** for each violation.
- You CANNOT change or add instance variables, and you CANNOT use **System.out** statements in this class. **-2 points** for each violation.

(g) **Timeslot class**

```
public class Timeslot implements Comparable<Timeslot> {
    private Date date;
    private Time time;

    @Override
    public String toString() { }

    @Override
    public int compareTo(Timeslot slot) { }

}
```

- You must implement the methods listed above. You must implement the Comparable Interface and implement the **compareTo()** method, or **lose 2 points** for each violation.
- You CANNOT change or add instance variables, and you CANNOT use **System.out** statements in this class, EXCEPT the testbed main(), **-2 points** for each violation.
- You MUST design the test cases to thoroughly test the **compareTo()** method. You must write a testbed main and implement the test cases. You must follow the instructions in the "Test Design" section in the "Software Project Coding Standard and Ground Rules. In the testbed main, you MUST write code to print out the test results to the console showing the test cases are passed or failed. The testbed main for this class **is worth 10 points**. Please use "//" comments to identify the test case numbers in the testbed main.

(h) **Patient class**

```
public class Patient implements Comparable<Patient> {
    private String fname;
    private String lname;
    private Date   dob;

    @Override
    public String toString() { }

    @Override
    public int compareTo(Patient patient) { }

}
```

- You must implement the methods listed above. You must implement the Comparable Interface and implement the **compareTo()** method, or **lose 2 points** for each violation.
- You CANNOT change or add instance variables, and you CANNOT use **System.out** statements in this class. **-2 points** for each violation.

(i) **Location class**

```
public enum Location { }
```

- You must use an enum class to define the 5 locations available for vaccination appointments. Use the county names as the constant names and define the zip codes and city names as the properties accordingly. See Lecture Note #2 for the syntax to define constants in the enum class.

8. You are required to **generate the Javadoc** after you properly commented your code. Your Javadoc must include the documentations for the constructors, private methods, and public methods of all Java classes. Generate the Javadoc in a single folder "**doc**" and include it in the zip file to be submitted to Canvas. **Please double check your Javadoc after you generated it** and make sure the descriptions are NOT EMPTY. You will lose points if any description in the Javadoc is empty. You will **lose 5 points** for not including the Javadoc.

Sample Input – the graders will be using the sample input to grade your project. They will run the transactions in batch with the same sequence shown below.

```
b
c
p
pz
pp
cp
q
B 4/1/2022 John Doe 12/1/2022 9:00 SOMERSET
B 3/1/2000 John Doe 12/1/2021 9:00 SOMERSET
B 12/1/1999 John Doe 12/1/2022 9:10 SOMERSET
B 12/1/1999 John Doe 12/1/2022 8:00 SOMERSET
B 12/1/1999 John Doe 12/1/2022 17:00 SOMERSET
B 12/1/1999 John Doe 12/1/2022 9:30 XXXXX
B 12/1/1999 John Doe 3/32/2022 9:45 SOMERSET
B 11/31/1999 John Doe 3/31/2022 9:45 SOMERSET
B 2/29/2011 John Doe 3/32/2022 9:45 SOMERSET
B 12/1/1999 John Doe 12/1/2022 9:45 SOMERSET
B 12/1/1999 John Doe 12/1/2022 9:45 SOMERSET
B 12/12/1989 John Doe 12/1/2022 9:45 SOMERSET
B 1/19/2000 Jane Doe 12/1/2022 9:45 union
B 3/31/1995 John Doe 2/28/2022 16:45 SOMERSET
B 1/15/2003 April March 3/2/2023 12:00 MORRIS
B 1/15/2003 April March 3/2/2022 12:00 MORRIS
B 7/31/1992 Chris Young 2/27/2022 11:15 Morris
B 2/29/2008 Roy Brooks 4/30/2022 10:30 middlesex
B 5/31/2000 Kate Lindsey 6/30/2022 13:30 mercer
B 11/1/1988 Duke Ellington 7/1/2022 14:15 mercer
B 3/29/2008 Roy Brooks 4/30/2022 15:15 middlesex
B 8/31/1978 Jane Doe 9/19/2022 9:15 union
B 8/31/1978 Jane Doe 9/19/2022 9:15 middlesex
B 8/31/1978 Jane Doe 7/19/2022 9:15 middlesex
P
C 8/31/1978 Jane Doe 7/19/2022 9:15 middlesex
C 8/31/1978 Jane Doe 7/19/2022 9:15 middlesex
C 8/31/1978 Jane Doe 1/19/2022 9:15 union
C 8/31/1978 Jane Doe 9/19/2022 9:15 union
P
B 8/31/1978 Jane Doe 7/19/2022 9:15 middlesex
B 8/31/1978 Jane Doe 9/19/2022 9:15 union
B 8/31/1978 Jane Doe 12/1/2022 12:15 somerset
B 1/1/1988 Duke Ellington 2/28/2022 14:30 somerset
PZ
PP
CP 8/31/1978 Jane Doe
P
B 1/15/2003 April March 5/5/2022 12:00 UNION
B 1/15/2003 April March 3/25/2022 16:00 middlesex
PP
q
Q
This line should not be processed.
```

Sample Output

```
Kiosk running. Ready to process transactions.

Invalid command!
Invalid command!
Invalid command!
Invalid command!
Invalid command!
Invalid command!
Invalid command!
Date of birth invalid -> it is a future date.
Appointment date invalid -> must be a future date.
Invalid appointment time! Must enter a time between 9:00 and 16:45 with a 15-minute interval.
Invalid appointment time! Must enter a time between 9:00 and 16:45 with a 15-minute interval.
Invalid appointment time! Must enter a time between 9:00 and 16:45 with a 15-minute interval.
Invalid location!
Invalid appointment date!
Invalid date of birth!
Invalid date of birth!
Appointment booked and added to the schedule.
Same appointment exists in the schedule.
Time slot has been taken at this location.
Appointment booked and added to the schedule.
Appointment booked and added to the schedule.
Invalid appointment date!
Appointment booked and added to the schedule.
Appointment booked and added to the schedule.
Appointment booked and added to the schedule.
Appointment booked and added to the schedule.
Appointment booked and added to the schedule.
Appointment booked and added to the schedule.
Appointment booked and added to the schedule.
Same patient cannot book an appointment with the same date.
Appointment booked and added to the schedule.

*list of appointments in the schedule*
John Doe, DOB: 12/1/1999, Appointment detail: 12/1/2022, 9:45, Bridgewater 08807, SOMERSET
Jane Doe, DOB: 1/19/2000, Appointment detail: 12/1/2022, 9:45, Union 07083, UNION
John Doe, DOB: 3/31/1995, Appointment detail: 2/28/2022, 16:45, Bridgewater 08807, SOMERSET
April March, DOB: 1/15/2003, Appointment detail: 3/2/2022, 12:00, Morristown 07960, MORRIS
Chris Young, DOB: 7/31/1992, Appointment detail: 2/27/2022, 11:15, Morristown 07960, MORRIS
Roy Brooks, DOB: 2/29/2008, Appointment detail: 4/30/2022, 10:30, Piscataway 08854, MIDDLESEX
Kate Lindsey, DOB: 5/31/2000, Appointment detail: 6/30/2022, 13:30, Princeton 08542, MERCER
Duke Ellington, DOB: 11/1/1988, Appointment detail: 7/1/2022, 14:15, Princeton 08542, MERCER
Roy Brooks, DOB: 3/29/2008, Appointment detail: 4/30/2022, 15:15, Piscataway 08854, MIDDLESEX
Jane Doe, DOB: 8/31/1978, Appointment detail: 9/19/2022, 9:15, Union 07083, UNION
Jane Doe, DOB: 8/31/1978, Appointment detail: 7/19/2022, 9:15, Piscataway 08854, MIDDLESEX
*end of schedule*

Appointment cancelled.
Not cancelled, appointment does not exist.
Not cancelled, appointment does not exist.
Appointment cancelled.

*list of appointments in the schedule*
John Doe, DOB: 12/1/1999, Appointment detail: 12/1/2022, 9:45, Bridgewater 08807, SOMERSET
Jane Doe, DOB: 1/19/2000, Appointment detail: 12/1/2022, 9:45, Union 07083, UNION
John Doe, DOB: 3/31/1995, Appointment detail: 2/28/2022, 16:45, Bridgewater 08807, SOMERSET
April March, DOB: 1/15/2003, Appointment detail: 3/2/2022, 12:00, Morristown 07960, MORRIS
Chris Young, DOB: 7/31/1992, Appointment detail: 2/27/2022, 11:15, Morristown 07960, MORRIS
Roy Brooks, DOB: 2/29/2008, Appointment detail: 4/30/2022, 10:30, Piscataway 08854, MIDDLESEX
Kate Lindsey, DOB: 5/31/2000, Appointment detail: 6/30/2022, 13:30, Princeton 08542, MERCER
Duke Ellington, DOB: 11/1/1988, Appointment detail: 7/1/2022, 14:15, Princeton 08542, MERCER
```

```
Roy Brooks, DOB: 3/29/2008, Appointment detail: 4/30/2022, 15:15, Piscataway 08854, MIDDLESEX
*end of schedule*

Appointment booked and added to the schedule.
Appointment booked and added to the schedule.
Appointment booked and added to the schedule.
Appointment booked and added to the schedule.

*list of appointments by zip and time slot.
Jane Doe, DOB: 8/31/1978, Appointment detail: 9/19/2022, 9:15, Union 07083, UNION
Jane Doe, DOB: 1/19/2000, Appointment detail: 12/1/2022, 9:45, Union 07083, UNION
Chris Young, DOB: 7/31/1992, Appointment detail: 2/27/2022, 11:15, Morristown 07960, MORRIS
April March, DOB: 1/15/2003, Appointment detail: 3/2/2022, 12:00, Morristown 07960, MORRIS
Kate Lindsey, DOB: 5/31/2000, Appointment detail: 6/30/2022, 13:30, Princeton 08542, MERCER
Duke Ellington, DOB: 11/1/1988, Appointment detail: 7/1/2022, 14:15, Princeton 08542, MERCER
Duke Ellington, DOB: 1/1/1988, Appointment detail: 2/28/2022, 14:30, Bridgewater 08807,
SOMERSET
John Doe, DOB: 3/31/1995, Appointment detail: 2/28/2022, 16:45, Bridgewater 08807, SOMERSET
John Doe, DOB: 12/1/1999, Appointment detail: 12/1/2022, 9:45, Bridgewater 08807, SOMERSET
Jane Doe, DOB: 8/31/1978, Appointment detail: 12/1/2022, 12:15, Bridgewater 08807, SOMERSET
Roy Brooks, DOB: 2/29/2008, Appointment detail: 4/30/2022, 10:30, Piscataway 08854, MIDDLESEX
Roy Brooks, DOB: 3/29/2008, Appointment detail: 4/30/2022, 15:15, Piscataway 08854, MIDDLESEX
Jane Doe, DOB: 8/31/1978, Appointment detail: 7/19/2022, 9:15, Piscataway 08854, MIDDLESEX
*end of schedule.


*list of appointments by patient.
Roy Brooks, DOB: 2/29/2008, Appointment detail: 4/30/2022, 10:30, Piscataway 08854, MIDDLESEX
Roy Brooks, DOB: 3/29/2008, Appointment detail: 4/30/2022, 15:15, Piscataway 08854, MIDDLESEX
Jane Doe, DOB: 8/31/1978, Appointment detail: 12/1/2022, 12:15, Bridgewater 08807, SOMERSET
Jane Doe, DOB: 8/31/1978, Appointment detail: 9/19/2022, 9:15, Union 07083, UNION
Jane Doe, DOB: 8/31/1978, Appointment detail: 7/19/2022, 9:15, Piscataway 08854, MIDDLESEX
Jane Doe, DOB: 1/19/2000, Appointment detail: 12/1/2022, 9:45, Union 07083, UNION
John Doe, DOB: 3/31/1995, Appointment detail: 2/28/2022, 16:45, Bridgewater 08807, SOMERSET
John Doe, DOB: 12/1/1999, Appointment detail: 12/1/2022, 9:45, Bridgewater 08807, SOMERSET
Duke Ellington, DOB: 1/1/1988, Appointment detail: 2/28/2022, 14:30, Bridgewater 08807,
SOMERSET
Duke Ellington, DOB: 11/1/1988, Appointment detail: 7/1/2022, 14:15, Princeton 08542, MERCER
Kate Lindsey, DOB: 5/31/2000, Appointment detail: 6/30/2022, 13:30, Princeton 08542, MERCER
April March, DOB: 1/15/2003, Appointment detail: 3/2/2022, 12:00, Morristown 07960, MORRIS
Chris Young, DOB: 7/31/1992, Appointment detail: 2/27/2022, 11:15, Morristown 07960, MORRIS
*end of list

All appointments for Jane Doe, DOB: 8/31/1978 have been cancelled.

*list of appointments in the schedule*
Roy Brooks, DOB: 2/29/2008, Appointment detail: 4/30/2022, 10:30, Piscataway 08854, MIDDLESEX
Roy Brooks, DOB: 3/29/2008, Appointment detail: 4/30/2022, 15:15, Piscataway 08854, MIDDLESEX
Jane Doe, DOB: 1/19/2000, Appointment detail: 12/1/2022, 9:45, Union 07083, UNION
John Doe, DOB: 3/31/1995, Appointment detail: 2/28/2022, 16:45, Bridgewater 08807, SOMERSET
John Doe, DOB: 12/1/1999, Appointment detail: 12/1/2022, 9:45, Bridgewater 08807, SOMERSET
Duke Ellington, DOB: 1/1/1988, Appointment detail: 2/28/2022, 14:30, Bridgewater 08807,
SOMERSET
Duke Ellington, DOB: 11/1/1988, Appointment detail: 7/1/2022, 14:15, Princeton 08542, MERCER
Kate Lindsey, DOB: 5/31/2000, Appointment detail: 6/30/2022, 13:30, Princeton 08542, MERCER
April March, DOB: 1/15/2003, Appointment detail: 3/2/2022, 12:00, Morristown 07960, MORRIS
Chris Young, DOB: 7/31/1992, Appointment detail: 2/27/2022, 11:15, Morristown 07960, MORRIS
*end of schedule*

Appointment booked and added to the schedule.
Appointment booked and added to the schedule.

*list of appointments by patient.
```

```
Roy Brooks, DOB: 2/29/2008, Appointment detail: 4/30/2022, 10:30, Piscataway 08854, MIDDLESEX
Roy Brooks, DOB: 3/29/2008, Appointment detail: 4/30/2022, 15:15, Piscataway 08854, MIDDLESEX
Jane Doe, DOB: 1/19/2000, Appointment detail: 12/1/2022, 9:45, Union 07083, UNION
John Doe, DOB: 3/31/1995, Appointment detail: 2/28/2022, 16:45, Bridgewater 08807, SOMERSET
John Doe, DOB: 12/1/1999, Appointment detail: 12/1/2022, 9:45, Bridgewater 08807, SOMERSET
Duke Ellington, DOB: 1/1/1988, Appointment detail: 2/28/2022, 14:30, Bridgewater 08807,
SOMERSET
Duke Ellington, DOB: 11/1/1988, Appointment detail: 7/1/2022, 14:15, Princeton 08542, MERCER
Kate Lindsey, DOB: 5/31/2000, Appointment detail: 6/30/2022, 13:30, Princeton 08542, MERCER
April March, DOB: 1/15/2003, Appointment detail: 3/2/2022, 12:00, Morristown 07960, MORRIS
April March, DOB: 1/15/2003, Appointment detail: 5/5/2022, 12:00, Union 07083, UNION
April March, DOB: 1/15/2003, Appointment detail: 3/25/2022, 16:00, Piscataway 08854, MIDDLESEX
Chris Young, DOB: 7/31/1992, Appointment detail: 2/27/2022, 11:15, Morristown 07960, MORRIS
*end of list

Invalid command!
Kiosk session ended.
```