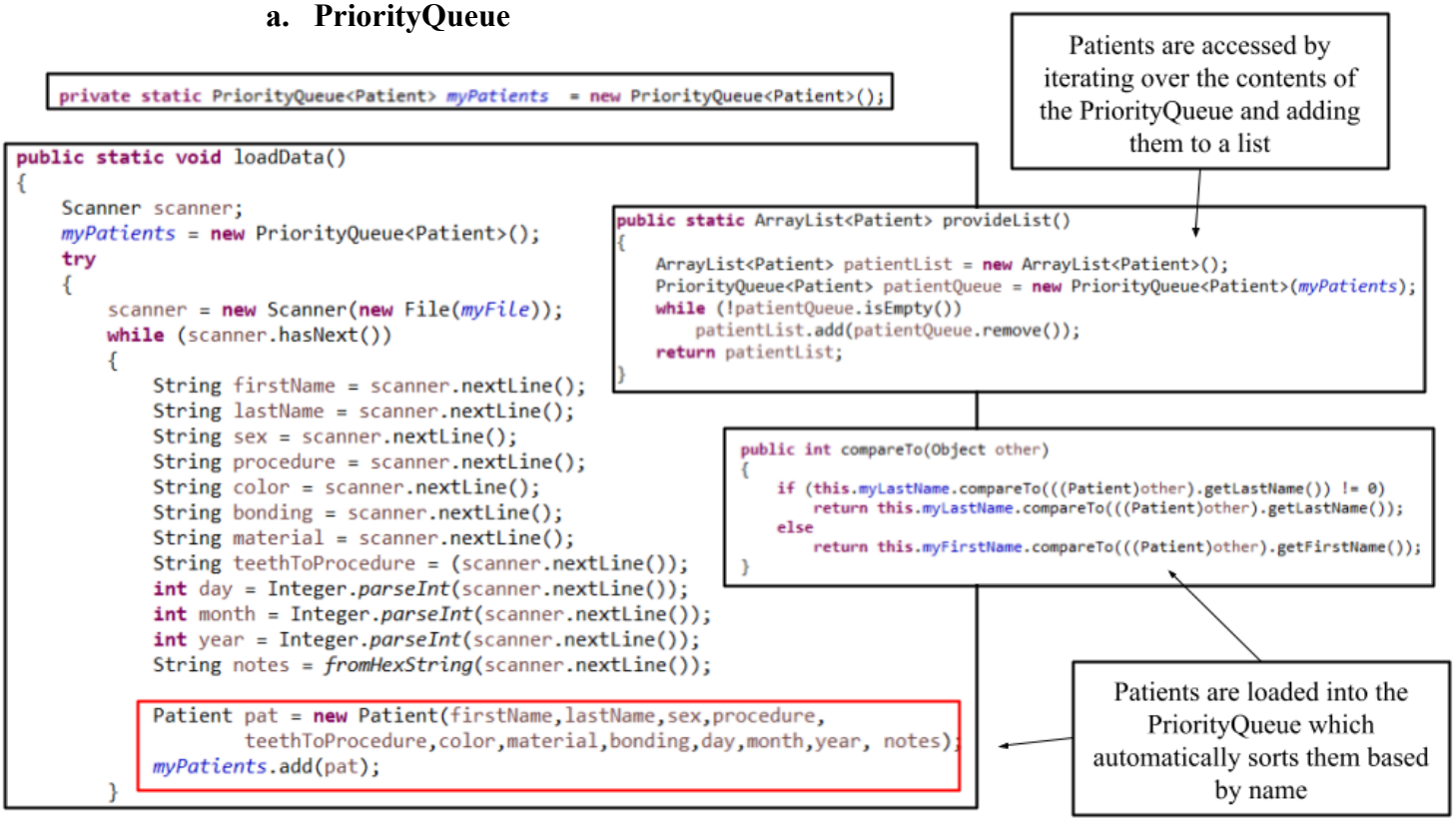
Criterion C: Development

Table of Contents:

1. Complex Data Types

a. PriorityQueue




```
private static PriorityQueue<Patient> myPatients  = new PriorityQueue<Patient>();
```

```
public static void loadData()
{
    Scanner scanner;
    myPatients = new PriorityQueue<Patient>();
    try
    {
        scanner = new Scanner(new File(myFile));
        while (scanner.hasNext())
        {
            String firstName = scanner.nextLine();
            String lastName = scanner.nextLine();
            String sex = scanner.nextLine();
            String procedure = scanner.nextLine();
            String color = scanner.nextLine();
            String bonding = scanner.nextLine();
            String material = scanner.nextLine();
            String teethToProcedure = (scanner.nextLine());
            int day = Integer.parseInt(scanner.nextLine());
            int month = Integer.parseInt(scanner.nextLine());
            int year = Integer.parseInt(scanner.nextLine());
            String notes = fromHexString(scanner.nextLine());

            Patient pat = new Patient(firstName,lastName,sex,procedure,
                    teethToProcedure,color,material,bonding,day,month,year, notes);
            myPatients.add(pat);
        }
```

```
public static ArrayList<Patient> provideList()
{
    ArrayList<Patient> patientList = new ArrayList<Patient>();
    PriorityQueue<Patient> patientQueue = new PriorityQueue<Patient>(myPatients);
    while (!patientQueue.isEmpty())
        patientList.add(patientQueue.remove());
    return patientList;
}
```

```
public int compareTo(Object other)
{
    if (this.myLastName.compareTo(((Patient)other).getLastName()) != 0)
        return this.myLastName.compareTo(((Patient)other).getLastName());
    else
        return this.myFirstName.compareTo(((Patient)other).getFirstName());
}
```

A PriorityQueue is used to store the data on all the patients due to how it automatically sorts objects using the compareTo() method of the data type specified. For a patient data manager, this feature adds convenience by having the PriorityQueue sort the patients alphabetically based on last name and first name. Additionally, the user needs to have the freedom to add as many or as few patients as they need into the program. Thankfully, PriorityQueues do not require a fixed length to be set on initialization and can change their size based on the amount of data. Finally, it is very simple to remove unwanted data from a PriorityQueue.

b. ArrayList

ArrayList are used in the main menu as opposed to PriorityQueues due to how effectively the get() method of the ArrayList works in tandem with selecting items from a JTable. When the user selects patients from the table, the index of the selected row can be used to easily access the Patient object at the corresponding index in the ArrayList, allowing for easy and efficient data retrieval.

2. 2D Array - Matrix

```
private void loadTable() {
    String[][] table = new String[patientList.size()][Patient.CATEGORIES.length];
    for (int i = 0; i < patientList.size(); i++) {
        Patient pat = patientList.get(i);
        String dateOfBirth = pat.getMonth() + "/" + pat.getDay() + "/" + pat.getYear();
        table[i] = new String[]{ pat.getLastName(), pat.getFirstName(), dateOfBirth, pat.getSex(),
                pat.getProcedure(),pat.getMaterial(),pat.getBonding(),pat.getColor() };

        modelTable = new DefaultTableModel(table, Patient.CATEGORIES) {

            @Override
            public boolean isCellEditable(int row, int column) {
                //all cells false
                return false;
            }
        };
    }
    tablePatient.setModel(modelTable);
}
```

Each row of the matrix is created by iterating over the patient data; it is then used to create the table

A two-dimensional array, otherwise known as a matrix, is used in the main menu to create the table, with the rows and columns of the matrix corresponding to the rows and columns of the table. Each cell of the matrix holds a string representing a piece of data for a patient which the table then displays.

Last Name	First Name	D.O.B	Sex	Procedure	Material	Bonding	Color
James	Andrews	5/4/1996	Male	Other	Flowable Composite	Adhesive	A2
Smith	Jane	4/1/2003	Female	Filling	Zirconia	Cement	A1
Smith	John	1/1/2001	Male	Crown	Flowable Composite	Adhesive	A3
Wilson	Andrew	7/3/2003	Male	Filling	Flowable Composite	Adhesive	A1

Each row holds the data for a single patient. The first column holds their last name, the second their first name, the third their date of birth, the fourth their sex, the fifth their procedure to be performed, the sixth the material to be used, the seventh the bonding to be used, and lastly the color of the material.

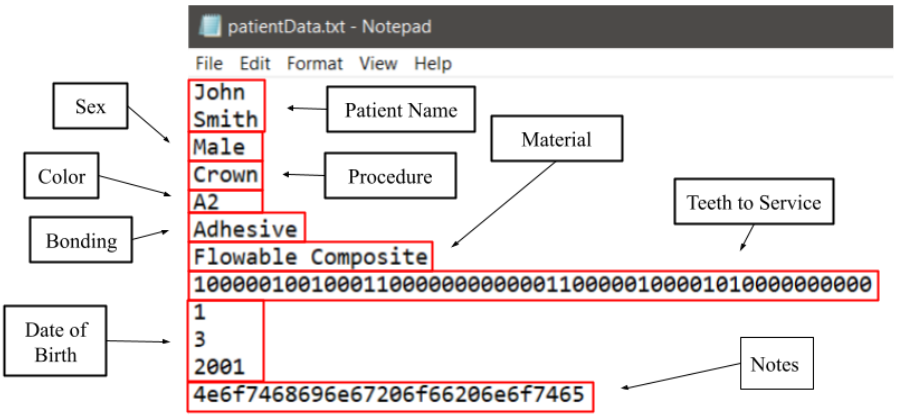
3. File Handling

a. Writing to a File

```
public static void saveData()
{
    //save patients
    Iterator<Patient> iter = myPatients.iterator();
    PrintWriter writer;
    try
    {
        writer = new PrintWriter(new File(myFile));
        while(iter.hasNext())
        {
            Patient pat = iter.next();
            System.out.println(pat);
            writer.print(pat.toString());
        }
        writer.close();
    }
    catch (Exception e)
    {
        System.out.println("Error: " + e.getMessage());
    }
    printList();
```

Patient's toString() method converts all of its data into single string that is written to file

```
public String toString()
{
    return myFirstName + "\n" + myLastName + "\n" + mySex + "\n" + myProcedure + "\n" +
           myColor + "\n" + myBonding + "\n" + myMaterial + "\n" + myTeethToProcedure + "\n"
           + myDay + "\n" + myMonth + "\n" + myYear + "\n" + Database.toHexString(myNotes) + "\n";
}
```



A PrintWriter object is used to write the patient data to a text file located in the same directory as the program. This allows the user to keep their data and any changes made between closing and reopening the program as well as transfer data using the text file. The text file itself is formatted as shown in the image, each line being a different piece of patient data. The teeth to be serviced is stored as a binary string, with the 1's being the teeth to be serviced.

```java
public static String fromHexString(String hex) {
  if (hex.length() == 0)
      return "";
  StringBuilder str = new StringBuilder();
  for (int i = 0; i < hex.length(); i+=2) {
    str.append((char) Integer.parseInt(hex.substring(i, i + 2), 16));
  }
  return str.toString();
}
```

```java
public static String toHexString(String text) {
    StringBuilder str = new StringBuilder();
    byte[] ba = text.getBytes(StandardCharsets.UTF_8);
    for(int i = 0; i < ba.length; i++)
        str.append(String.format("%02x", ba[i]));
    if (str.length() == 0)
        return "";
    else
        return str.toString();
 }
```

The notes on a patient are converted to hexadecimal so that it may be stored as a single string so that the program does not need to account for new line characters when reading the file. The hexadecimal-to-string and string-to-hexadecimal conversion code was adapted from code posted by Stack Overflow user *jordeu* on April 11, 2012[1]

[1] jordeu. "Converting a String to Hexadecimal in Java." *Stack Overflow*, 11 Apr. 2012, https://stackoverflow.com/questions/923863/converting-a-string-to-hexadecimal-in-java.

b. Reading from a File

```
public static void loadData()
{
    Scanner scanner;
    myPatients = new PriorityQueue<Patient>();
    try
    {
        scanner = new Scanner(new File(myFile));
        while (scanner.hasNext())
        {
            String firstName = scanner.nextLine();
            String lastName = scanner.nextLine();
            String sex = scanner.nextLine();
            String procedure = scanner.nextLine();
            String color = scanner.nextLine();
            String bonding = scanner.nextLine();
            String material = scanner.nextLine();
            String teethToProcedure = (scanner.nextLine());
            int day = Integer.parseInt(scanner.nextLine());
            int month = Integer.parseInt(scanner.nextLine());
            int year = Integer.parseInt(scanner.nextLine());
            String notes = fromHexString(scanner.nextLine());

            Patient pat = new Patient(firstName,lastName,sex,procedure,
                    teethToProcedure,color,material,bonding,day,month,year, notes);
            myPatients.add(pat);
        }
        scanner.close();
    }
    catch (Exception e)
    {
        System.out.println("Error: " + e.getMessage());
    }
}
```

Scanner reads the text file line by line

Some data needs to be parsed/converted

A new Patient is created using the data and added

A Scanner object is used to read patient data stored on a text file so that it can be converted into a series of Patient objects for the program to use. Whenever the program is opened, it automatically runs the loadData() method so that any previously saved data is redisplayed for the user to view and modify.

c. Image Handling

```
public static FileNameExtensionFilter filter = new FileNameExtensionFilter("Image File","png","jpg","jfif","jpeg","gif");
```

```
//add image for user
if (photoChooser.getSelectedFile() != null)
    addImage(pat);
```

```
photoChooser = new JFileChooser();
photoChooser.addChoosableFileFilter(Database.filter);
photoChooser.setAcceptAllFileFilterUsed(false);
```

```
//adds image for patient given selected image file and patient name & DOB
private void addImage(Patient pat)
{
    String header = pat.getLastName() + pat.getFirstName() + pat.getMonth() + pat.getDay() + pat.getYear();
    String path = photoChooser.getSelectedFile().getPath();
    String fileType = path.substring(path.indexOf("."));
    String newPath = "TempPics/" + header + fileType;
    System.out.println(newPath);

    try {
        Files.copy(Paths.get(path), Paths.get(newPath), StandardCopyOption.REPLACE_EXISTING);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

The image files are named based on the patient's name and date of birth

Copies original image file to folder of patient images

A JFileChooser is used to let the user select images for patients under the formats specified in the FileNameExtensionFilter containing all the most common image file formats. The image file is then copied into a folder of patient images and renamed using their name and date of birth.

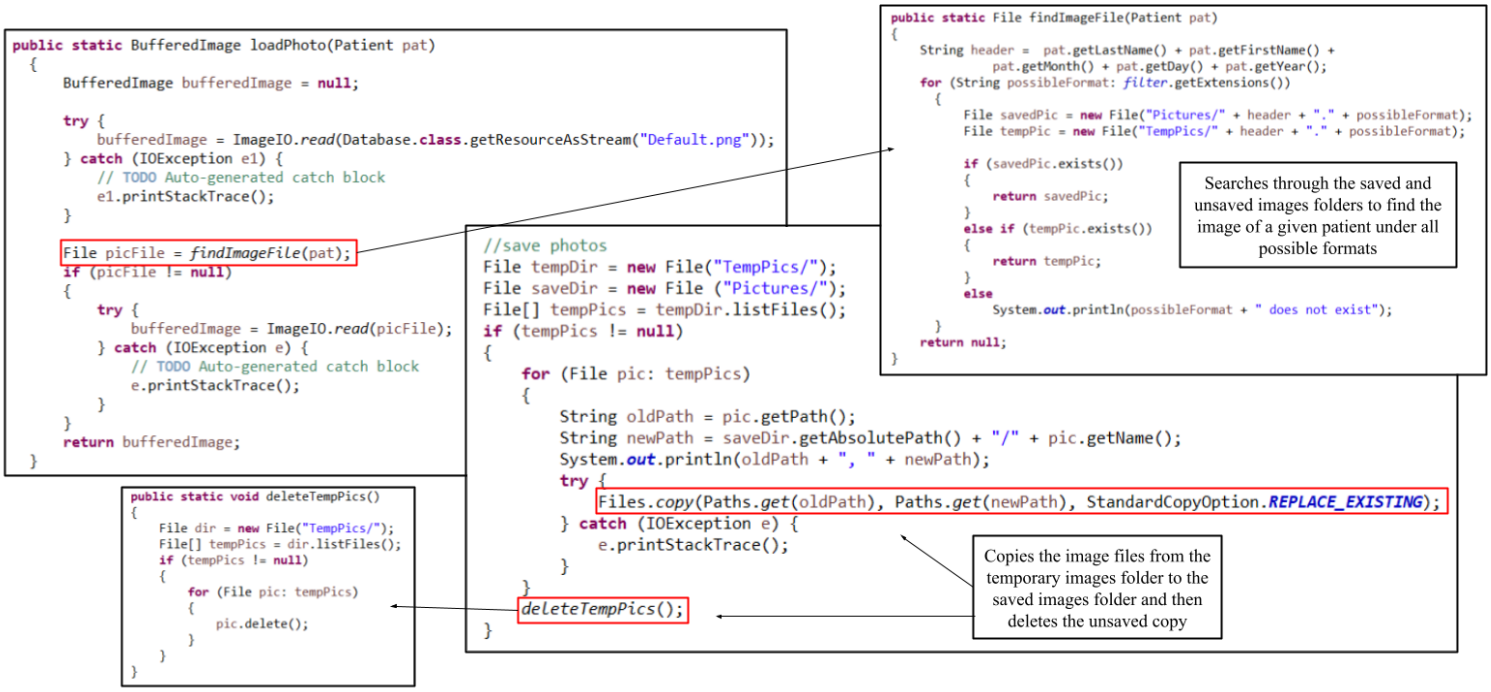


Image files for the patients can then be loaded into the program by searching the image folders for a patient's image file and then creating a BufferedImage object using the file so that it can be displayed. Image files are also saved by moving them from a folder of image files to be deleted when the program closes to a folder for image files to be retained between uses of the program.

4. Error Handling

a. Try/Catch

```
public static BufferedImage loadPhoto(Patient pat)
 {
     BufferedImage bufferedImage = null;

     try {
         bufferedImage = ImageIO.read(Database.class.getResourceAsStream("Default.png"));
     } catch (IOException e1) {
         // TODO Auto-generated catch block
         e1.printStackTrace();
     }

     File picFile = findImageFile(pat);
     if (picFile != null)
     {
         try {
             bufferedImage = ImageIO.read(picFile);
         } catch (IOException e) {
             // TODO Auto-generated catch block
             e.printStackTrace();
         }
     }
     return bufferedImage;
 }
```

```
File tempDir = new File("TempPics/");
File saveDir = new File ("Pictures/");
File[] tempPics = tempDir.listFiles();
if (tempPics != null)
{
    for (File pic: tempPics)
    {
        String oldPath = pic.getPath();
        String newPath = saveDir.getAbsolutePath() + "/" + pic.getName();
        System.out.println(oldPath + ", " + newPath);
        try {
            Files.copy(Paths.get(oldPath), Paths.get(newPath), StandardCopyOption.REPLACE_EXISTING);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Because of the extensive amount of image file handling employed in the program, many try/catch blocks are used because many of these operations such as reading an image file using ImageIO's read() method in order to initialize a BufferedImage object and copying an image file to a new location using Files's copy() method require it. This is in order to deal with errors that may arise from these image files or the specified file path not existing, for example.

b. Handling Invalid Values

```
if (firstName.equals("First Name") || lastName.equals("Last Name")
        || comboBoxSex.getSelectedIndex() == 0 || comboBoxProcedure.getSelectedIndex() == 0
        || comboBoxColor.getSelectedIndex() == 0 || comboBoxBonding.getSelectedIndex() == 0
        || comboBoxMaterial.getSelectedIndex() == 0)
{
    JOptionPane.showMessageDialog(getRootPane(),
            "       Error: Empty or invalid fields", "Invalid fields", JOptionPane.ERROR_MESSAGE);
    return false;
}
```

The program checks to see if a user has inputted all of the necessary data for a patient by checking each JTextField and JComboBox to see if each still contains their default values, meaning the user has not selected an option. If a field is invalid, an error message is displayed.

5. External Libraries

```
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.PriorityQueue;
import java.util.Scanner;
```

```
import java.awt.AWTEvent;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.border.EmptyBorder;
import javax.swing.table.DefaultTableModel;
import java.awt.Toolkit;
import javax.swing.JLabel;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import java.awt.Font;
```

The code for this program makes extensive use of external libraries in order to take advantage of many pre-made features. The left picture displays some of the imported libraries from the Database class such as java.io and java.nio for file handling as well as java.util for reading input and data structures such as ArrayList and PriorityQueue. The right picture displays some of the imported libraries from the MainMenu class such as java.awt and javax.swing used to create the program's GUI such as the buttons and table.

6. Encapsulation

```
//////////////////////////      setter methods   ////////////////////////////
public void setFirstName(String firstName) {myFirstName = firstName;}

public void setLastName(String lastName) {myLastName = lastName;}

public void setSex(String sex) {mySex = sex;}

public void setProcedure(String procedure) {myProcedure = procedure;}

public void setColor(String color) {myColor = color;}

public void setMaterial(String material) {myMaterial = material;}

public void setBonding(String bonding) {myBonding = bonding;}

public void setTeethToProcedure(String teethToProcedure) {myTeethToProcedure = teethToProcedure;}

public void setDay(int day) {myDay = day;}

public void setMonth(int month) {myMonth = month;}

public void setYear(int year) {myYear = year;}

public void setNotes(String notes) {myNotes = notes;}
```

```
//basic info
private String myFirstName;
private String myLastName;
private String mySex;

//DOB info
private int myDay;
private int myMonth;
private int myYear;

//procedure info
private String myProcedure;
private String myTeethToProcedure;
private String myColor;
private String myMaterial;
private String myBonding;

//other
private String myNotes;
```

```
////////////////////    getter methods  ////////////////
public String getFirstName() {return myFirstName;}

public String getLastName() {return myLastName;}

public String getSex()  {return mySex;}

public String getProcedure() {return myProcedure;}

public String getColor() {return myColor;}

public String getMaterial() {return myMaterial;}

public String getBonding() {return myBonding;}

public String getTeethToProcedure() {return myTeethToProcedure;}

public int getDay() {return myDay;}

public int getMonth() {return myMonth;}

public int getYear() {return myYear;}

public String getNotes() {return myNotes;}
```

The Patient class is an example of encapsulation within the program. All of the variables such as the patient's name, sex, and procedure information are kept private and can only be accessed and modified through the public access methods. Each variable has its own setter and getter methods.

7. Inheritance

```
public class Window extends JFrame {

    //default constructor for a window
    public Window()
    {
        setResizable(false);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResource("icon.png")));
    }
}
```

All of the classes below inherit the Window class's default constructor

```
public class MainMenu extends Window
```

```
public class PatientInfoWindow extends Window
```

```
public class TeethSelectionWindow extends Window
```

```
public class AddEditPatientWindow extends Window
```

The program most obviously makes use of inheritance through all windows extending the JFrame class containing all of the necessary methods and attributes to create a working GUI. However, to make sure that all of the windows inherit certain attributes such as not being resizable and having the same icon image, all of the windows extend the Window class, inheriting a constructor that sets all of these components automatically.

8. Searching

```
private void searchTable()
{
    resetTable();

    patientList = Database.provideList();

    ArrayList<Patient> wantedList = new ArrayList<Patient>();

    for (Patient pat : patientList)
    {
        if  (fitsFilter(pat))
                wantedList.add(pat);
    }

    patientList = wantedList;

    loadTable();
}
```

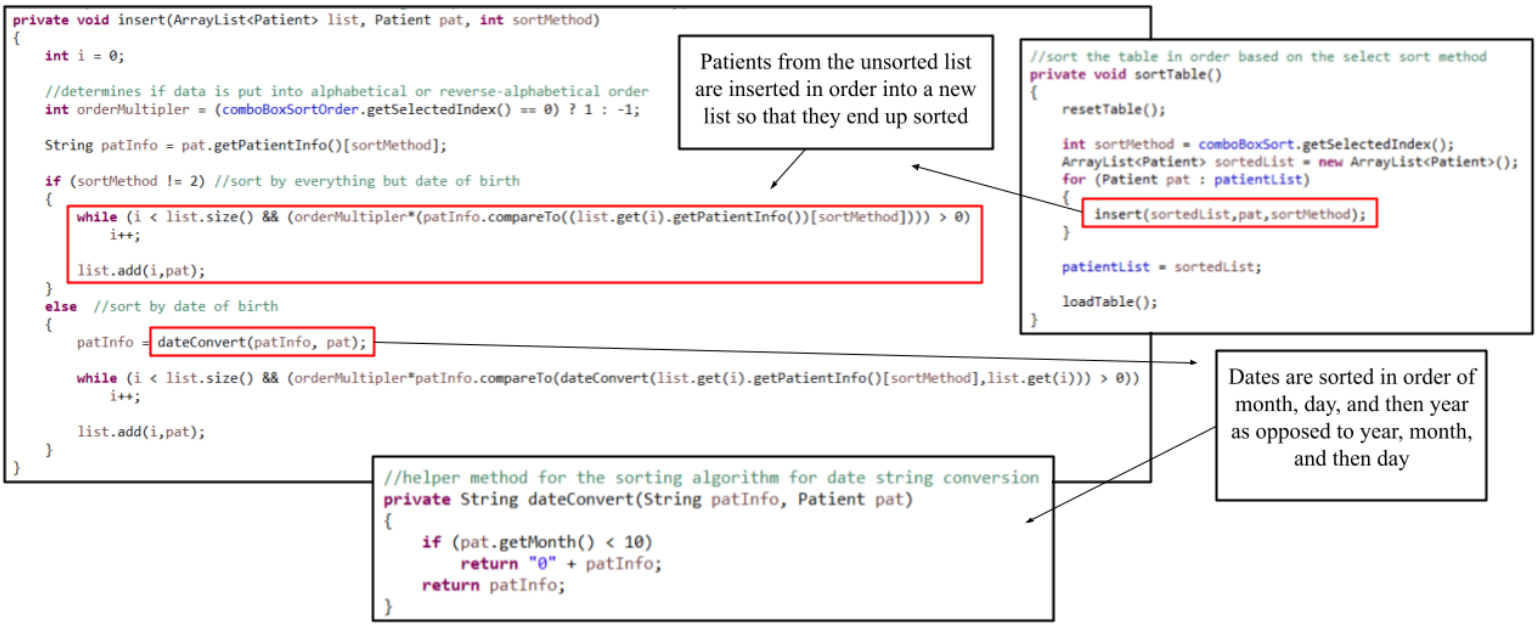
Each patient that satisfies all of if-statements in the fitsFilter() method are added to the sorted list

```
private boolean fitsFilter(Patient pat)
{
    String firstName = searchFirstName.getText();
    String lastName = searchLastName.getText();

    return ((firstName.equals("") || firstName.equals("Enter here...")|| ((pat.getFirstName().toLowerCase()).indexOf(firstName.toLowerCase()) == 0)) &&
            (lastName.equals("") || lastName.equals("Enter here...") || ((pat.getLastName().toLowerCase()).indexOf(lastName.toLowerCase()) == 0)) &&
            (searchSex.getSelectedIndex() == 0 || ((String)searchSex.getSelectedItem()).equals(pat.getSex())) &&
            (searchProcedure.getSelectedIndex() == 0 || ((String)searchProcedure.getSelectedItem()).equals(pat.getProcedure())) &&
            (searchMaterial.getSelectedIndex() == 0 || ((String)searchMaterial.getSelectedItem()).equals(pat.getMaterial())) &&
            (searchBonding.getSelectedIndex() == 0 || ((String)searchBonding.getSelectedItem()).equals(pat.getBonding())) &&
            (searchColor.getSelectedIndex() == 0 || ((String)searchColor.getSelectedItem()).equals(pat.getColor())));
}
```

The main menu's list of patients is traversed sequentially to search for all the patients that fit the search criteria specified by the user using the GUI and added to a new list of patients that is then displayed in the table. This allows the user to search for specific patients based on specific pieces of information.

9. Sorting




```
private void insert(ArrayList<Patient> list, Patient pat, int sortMethod)
{
    int i = 0;

    //determines if data is put into alphabetical or reverse-alphabetical order
    int orderMultipler = (comboBoxSortOrder.getSelectedIndex() == 0) ? 1 : -1;

    String patInfo = pat.getPatientInfo()[sortMethod];

    if (sortMethod != 2) //sort by everything but date of birth
    {
        while (i < list.size() && (orderMultipler*(patInfo.compareTo((list.get(i).getPatientInfo())[sortMethod]))) > 0)
            i++;

        list.add(i,pat);
    }
    else  //sort by date of birth
    {
        patInfo = dateConvert(patInfo, pat);

        while (i < list.size() && (orderMultipler*patInfo.compareTo(dateConvert(list.get(i).getPatientInfo()[sortMethod],list.get(i))) > 0))
            i++;

        list.add(i,pat);
    }
}
```

```
//sort the table in order based on the select sort method
private void sortTable()
{
    resetTable();

    int sortMethod = comboBoxSort.getSelectedIndex();
    ArrayList<Patient> sortedList = new ArrayList<Patient>();
    for (Patient pat : patientList)
    {
        insert(sortedList,pat,sortMethod);
    }

    patientList = sortedList;

    loadTable();
}
```

```
//helper method for the sorting algorithm for date string conversion
private String dateConvert(String patInfo, Patient pat)
{
    if (pat.getMonth() < 10)
        return "0" + patInfo;
    return patInfo;
}
```

Patients are sorted by taking the unsorted list in the main menu class and iterating over its contents. Based on the sorting method chosen by the user in the GUI (by name, date of birth, procedure, etc.), the contents are inserted into a new list in order. The main menu then stores the newly sorted list of patients and displays it to the table.

Word Count: 1108

Works Cited

jordeu. “Converting a String to Hexadecimal in Java.” *Stack Overflow*, 11 Apr. 2012, https://stackoverflow.com/questions/923863/converting-a-string-to-hexadecimal-in-java.