# Criterion C: Development

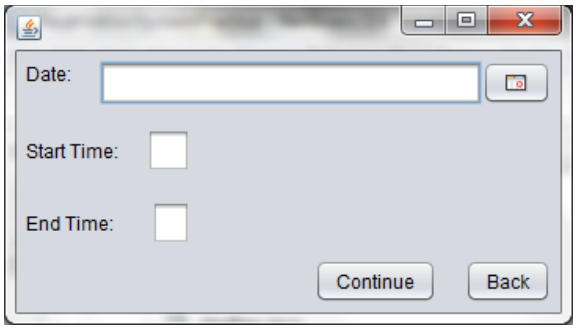
Techniques Used:

1. Graphical User interface (GUI)
2. Methods
3. Variables
4. Serialization (algorithmic thinking)
5. Importing into Combo Box
6. Cancel Reservation (algorithmic thinking)
7. Check for availability (algorithmic thinking)

## 1) Graphical User Interface (GUI):-

The graphical user interfaces that have been created are very user friendly, and have features that enhance the program's usability and make it more attractive to users such as:

- "Continue" and "Back" buttons that shift between GUIs, enhancing usability.
- As requested by the user, the initial reservation system screen contains an image of the company's logo, increasing the system's appeal.
- A date chooser button, which allows the user to click the date to enter it, instead of entering it in manually.
- Exit button that closes the program and exits the system.


Creating the GUIs was simplified by NetBeans, since it was very easy to design the GUIs on NetBeans, as it automatically created the Group Layout.

## 2) Methods:-

There were different methods for different buttons, in order to give functionality to these buttons. For example, the "Exit" button contained the following code:

```
   private void jButton4ActionPerformed(java.awt.event.ActionEvent evt)
{
      System.exit(0);
}
```

Similarly, all the “Back” buttons were given the following code or something similar, based on the previous screen:

```
   private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{
      ResSys ressys=new ResSys();
      ressys.setVisible(true);
      this.setVisible(false);
}
```

**3) Variables:-**

Local Variables

The local variables initialized in the program were the swing GUI components, which were declared in the method initComponents(). The text of these components/local variables were set in the same method. For example, in the Reservation System screen:

```
private void initComponents()
{
      NewResButton = new javax.swing.JButton();

      NewResButton.setText("New Reservation");
      NewResButton.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            NewResButtonActionPerformed(evt);
```

```
        }
    });
}
```

In the above piece of code, the “NewResButton” is initializaed as a JButton, its text is set as “New Reservation”, and an action listener is used to define that the code defined later should be implemented when the button is clicked.

Instance Variables

The swing GUI components were declared as instance variables outside of any method. For example, in the Reservation System screen, the instance variables were declared as follows:

```
private javax.swing.JButton NewResButton;
private javax.swing.JButton ModResButton;
private javax.swing.JButton CanResButton;
private javax.swing.JButton ExitButton;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private com.toedter.calendar.JDateChooser jDateChooser1;
```

Also, for serialization to take place, the following variables were declared as instance variables to be used in the serialization code to store values:

```
public String strField;     //Date
public int intField;        //Start Time
public int intField2;       //End Time
public String strField2;    //Name
```

**4) Serialization (algorithmic thinking):-**

Serialization is defined as “the process of translating data structures or object state into a format that can be stored and reconstructed later in the same or another computer environment” (“Serialization”). Serializing the GUI components was a simple job, but they also had to be reconstructed if the user wanted to modify a reservation. Hence, a new class, named

"NR" was created. This class contained variables which would store the values entered by the user for variables such as date, start time, end time, and name, and would serialize them.

The serialized files were saved in a file called "MyObject.ser". However, whenever the data was obtained from this file using "Modify Reservation", a NullPointerException error kept arising in the code. After attempting to debug the program, I realized that the error was arising due to the fact that, every time a new reservation was made, the data of the new reservation replaced the existing data of the previous reservation, hence causing the error. Hence, I decided to name the file differently. Every time a new user inputs the reservation data, the file is saved as "NameOfUser.ser". For example, if I make a reservation, the file would save as "Puneet.ser".

However, this would create another problem, since a single user would not be able to create a second reservation until his previous reservation was completed. Hence, in this system, each user is allowed only one reservation, which might make the system more efficient and reduce wastage of rooms, since employees will be more careful about their reservations.

**5) Importing into Combo Box:-**

Originally, the system was meant to be a room reservation system for several conference rooms. However, due to time constraints and the complexity of the issue, the client and I collaborated to decide that the system should be programmed for only room. This system could be used as a template for a future room reservation system for several rooms, hence displaying the extensibility of the product.

**6) Cancelling Reservation (algorithmic thinking):-**

Initially this aspect was very difficult to complete since I was unfamiliar with serialization and the handling of files through java. However, after some presence of mind, I decided to request the user to input his/her name, and to input the data from the file associated with the name entered by the user. Then, I would delete this file using the method ".delete()". The code behind this thinking is show below:

```
File file = new File(NameText.getText().concat(".ser")); //MyObject.ser");
    ObjectInputStream in;
    try{
       in = new
       ObjectInputStream(new FileInputStream(file));
       NameText.setText(res.strField2);
```

```
        System.out.println(res.strField2);
        in.close();
        file.delete();
    }
      catch(Exception e){
    e.printStackTrace();
}
```

**7) Clashing of reservations (algorithmic thinking):-**

One question that came up during my discussion with the client about the product was, “How do we prevent a conflict of reservations?” For this purpose, two array lists were created named “hours” and “dates”, which stored the times and dates of all the reservations. An attempt was made to perform a check for whether or not a new reservation being made clashed with an existing reservation, using the following code:

```
for(int i=0;i<res.dates.size();i++)
      {
         for(int j=0;j<res.hours.size();j++)
         {
            if(res.dates.get(i).equals(res.strField))
            {
               if(res.hours.get(j)==(res.intField))
               {
                  RNA rna=new RNA();
                  rna.setVisible(true);
                  this.setVisible(false);
               }
            }
```

```
    }
}
res.dates.add(res.strField);
res.hours.add(res.intField);
```

However, this code was not able to produce the expected result, and hence, I was unable to perform a check for the clashing of reservations.

Word count: 777