

Criterion C: Design

Techniques Used:

- *Page 1: Product Structure*
- *Page 2: Algorithmic thinking/Database structure*
- *Page 4: Listeners*
- *Page 5: Graphical Interface*
- *Page 8: File Handling*
- *Page 10: Error Handling*

Product Structure:

Most the overall structure of the product is the same as stated in criterion A. The main menu window that the program launches allows the client to go into either male or female athlete windows. From there, they are prompted add athletes into the database and then return to main menu. Now they can add times for each individual athlete for any chosen distance on any chosen date. All the data shall be saved into separate .csv files for each distance and date. From this point, they must return to either male or female windows to check times for each individual athlete on a chosen date. The output will be displayed in table form and will contain all the data for that chosen date, as well as athletes fastest times.

Some of the techniques used are: (these techniques will be elaborated on further in down)

- **Polymorphism:** men athlete/women athlete class referencing to individual check class.
- **Array Lists:** Storing dates and distances data.
- **Text Fields:** Reading the keyboard based input of user.
- **Error Handling:** Implementation of effective try-catch error handling.
- **Buttons:** responding to user clicks.
- **Reading/Writing files:** Inputting, retrieving and outputting of data into internal files

Algorithmic Thinking/Database Structure:

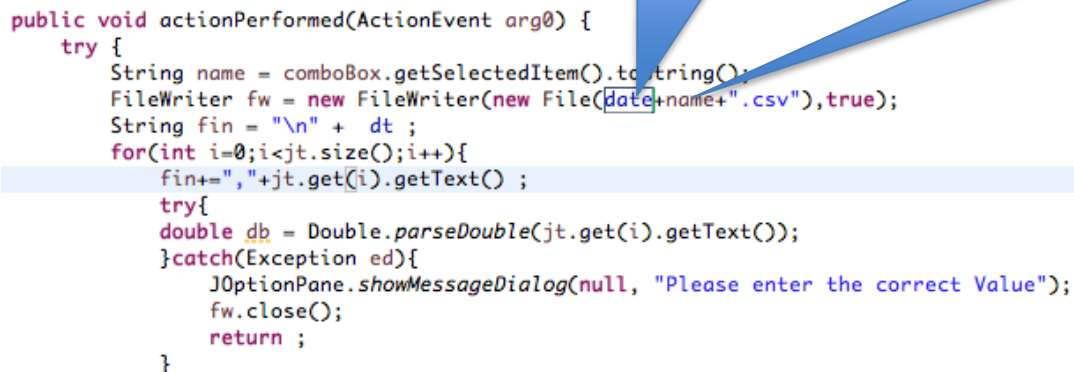
1. The database Structure is established through adding athletes and saving them to .CSV file from which values specific to them could be saved. This is achieved through the Adding athlete algorithm.

```
JButton btnAdd = new JButton("Add");
btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        File f= new File("per.csv");
        File fnew = new File(textField+".csv");
        try {
            FileWriter fw = new FileWriter(f,true);
            String name = textField.getText();
            fw.write(name+", "+gender+",true\n");
            fw.close();
            JOptionPane.showMessageDialog(null, name+ "" +"Athlete added");
            frame.setVisible(false);
            new MainMenu().frame.setVisible(true);
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }
});
```

Universal .CSV file,
which saves all athletes
into the same database

Figure 1: Illustrates code for Adding Athlete Algorithm

- Figure 1 illustrates the first action listener, which was created to add athletes into the database. Based on the gender, it saves the name in the format: (name+", "+gender+",true\n") into a .csv file.
2. Once Athletes have been added into database, specific and unique values can be assigned to them.



```

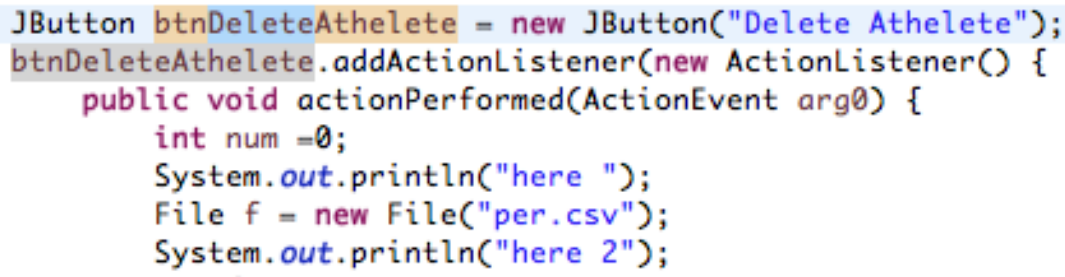
public void actionPerformed(ActionEvent arg0) {
    try {
        String name = comboBox.getSelectedItem().toString();
        FileWriter fw = new FileWriter(new File(date+name+".csv"),true);
        String fin = "\n" + dt ;
        for(int i=0;i<jt.size();i++){
            fin+=","+jt.get(i).getText() ;
            try{
                double db = Double.parseDouble(jt.get(i).getText());
            }catch(Exception ed){
                JOptionPane.showMessageDialog(null, "Please enter the correct Value");
            }
            fw.close();
            return ;
        }
    }
}

```

Figure 2: Portion of code that allows for individual entering of times

- Figure 2 illustrates the algorithm that is used to populate the athlete's profiles with saved times data.

3. Database Structure is reinforced by the ability to delete athletes and all of their corresponding values from the "per.csv" universal file.



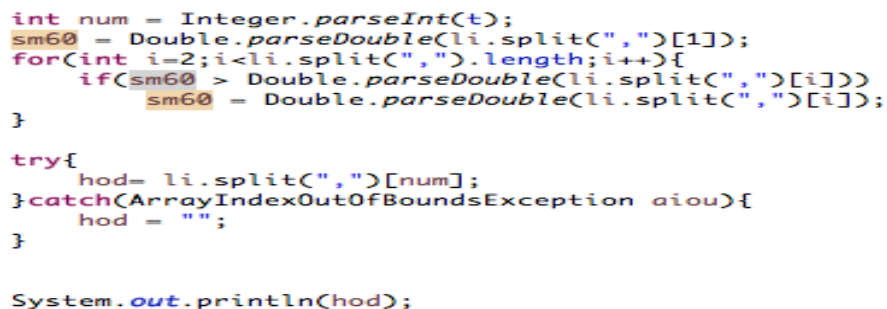
```

JButton btnDeleteAthelete = new JButton("Delete Athelete");
btnDeleteAthelete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        int num =0;
        System.out.println("here ");
        File f = new File("per.csv");
        System.out.println("here 2");
    }
}

```

Figure 3: Portion of code that allows for the deletion of an individuals profile and all corresponding values.

4. Once athletes times have been inputted, they can be viewed. The fastest time ran should always show up as well.



```

int num = Integer.parseInt(t);
sm60 = Double.parseDouble(li.split(",")[1]);
for(int i=2;i<li.split(",").length;i++){
    if(sm60 > Double.parseDouble(li.split(",")[i])){
        sm60 = Double.parseDouble(li.split(",")[i]);
    }
}
try{
    hod= li.split(",")[num];
}catch(ArrayIndexOutOfBoundsException aiou){
    hod = "";
}
System.out.println(hod);

```

Figure 4 illustrates the algorithm that detects and changes time in "fastest" table. This the code scans times for the 60m distance.

Figure 4: Algorithm that scans for whether fastest time has been entered.

Listeners

Being a graphics driven program, the foundation of the product is associated with the GUI design elements. Thus, all the data is inputted through the graphical interfaces, this means that processing/storing all the input was done through Listeners.

```
JButton btnAdd = new JButton("Add");
btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        File f= new File("per.csv");
        File fnew = new File(textField+".csv");
        try {
            FileWriter fw = new FileWriter(f,true);
            String name = textField.getText();
            fw.write(name+","+gender+",true\n");
            fw.close();
            JOptionPane.showMessageDialog(null, name+ " "+"Athlete added");
            frame.setVisible(false);
            new MainMenu().frame.setVisible(true);
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }
});
```

Figure 5: Adding Action Listener Athlete

```
JButton btnDeleteAthelete = new JButton("Delete Athelete");
btnDeleteAthelete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        int num =0;
        System.out.println("here ");
        File f = new File("per.csv");
        System.out.println("here 2");
    }
});
```

Figure 6:Deleting Athletes Action Listener

```
btnSaveTimes.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        try {
            String name = comboBox.getSelectedItemAt().toString();
            FileWriter fw = new FileWriter(new File("date"+name+".csv"),true);
            String fin = "\n" + dt ;
            for(int i=0;i<jt.size();i++){
                fin+=","+jt.get(i).getText() ;
            }
            fw.write(fin);
            fw.close();
        }
    }
});
```

Figure 7: code for adding times to athlete's profile

```
JButton btnNext = new JButton("Next ");
btnNext.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        String month = comboBox.getSelectedItemAt().toString();
        String day = comboBox_1.getSelectedItemAt().toString();
        frame.setVisible(false);
        addTimes ad = new addTimes(month+"/"+day +"/2017");
        ad.frame.setVisible(true);
    }
});
```

Figure 8: Example of one of many Action Listeners for buttons. This is for the “next button” on the add times window

Graphical Interface:

Most of the basic GUI components (buttons, Textfields, panels) were created through the Window Builder Plugin in Eclipse. Because most of GUI was thoroughly explained in the Design Criterion, I shall focus on:

The complex elements of the graphical interface were done manually:

1. Personalization of buttons on the main menu window: colour/font/location was done manually.

```
btnVarsityM.setForeground(new Color(255, 255, 255));  
btnVarsityM.setFont(new Font("Serif", Font.ITALIC, 13));  
btnVarsityM.setForeground(Color.blue);  
btnVarsityM.setBackground(Color.DARK_GRAY);  
btnVarsityM.setBounds(372, 191, 129, 23);  
panel.add(btnVarsityM);
```

Manual
Input of
font
type/colour
/location

Figure 9: Illustrating button personalization

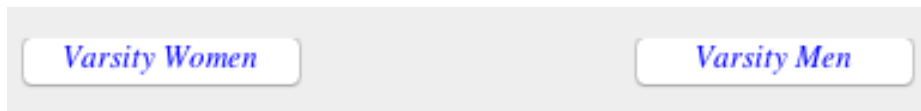


Figure 10: personalized Buttons on main menu.

2. Creating Combo-boxes within the program: Combo-boxes were created through importing the “`import javax.swing.JComboBox;`”.

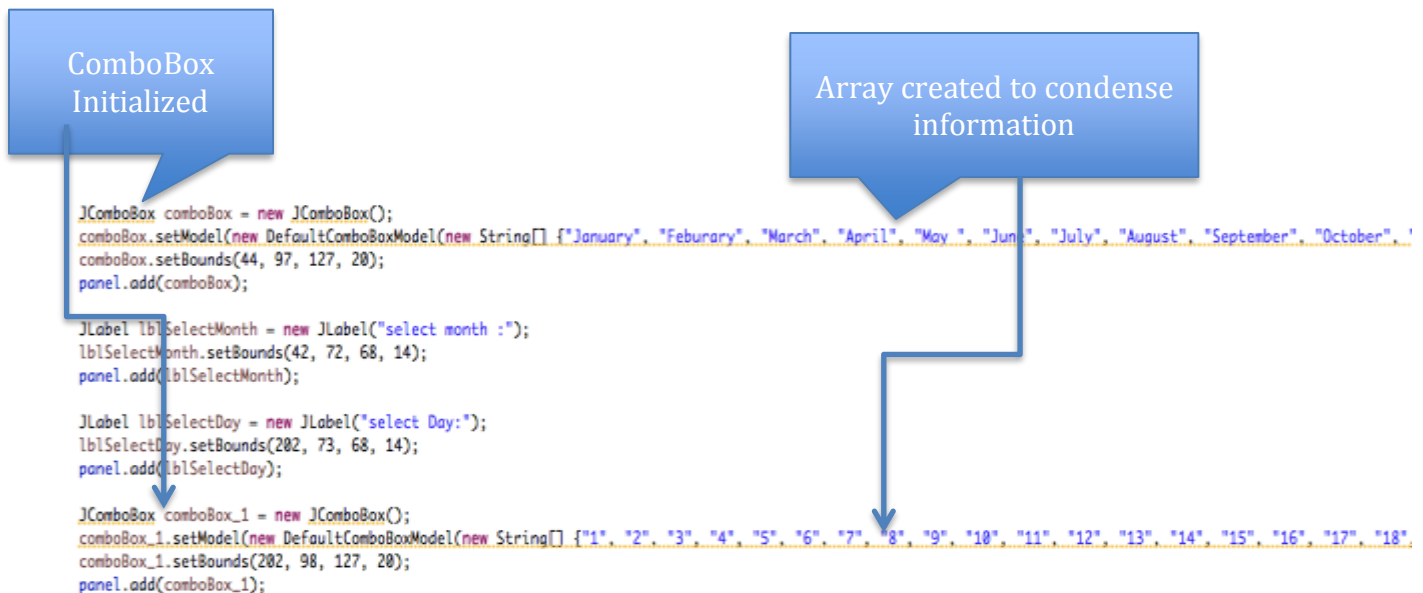


Figure 11: the code for initializing the arrays used to populate the “Date” comboboxes

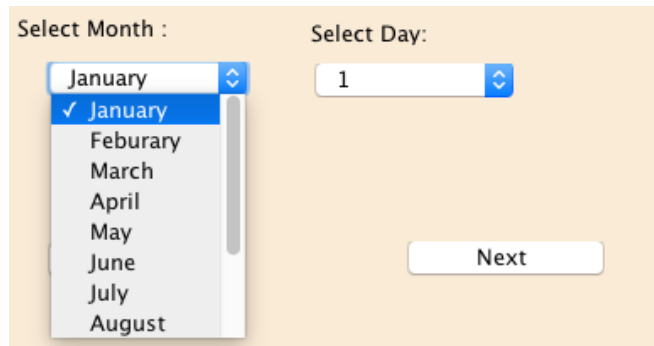


Figure 12: Graphical visualization of combo-box from figure 11

```
JComboBox comboBox = new JComboBox();  
comboBox.setModel(new DefaultComboBoxModel(new String[] {"60 meter", "100 meter", "150 meter", "200 meter", "300 meter", "400 meter"});  
comboBox.setBounds(20, 91, 260, 20);  
panel.add(comboBox);  
|
```

Figure 13: The code for initializing the arrays used to populate the “distances” combo box

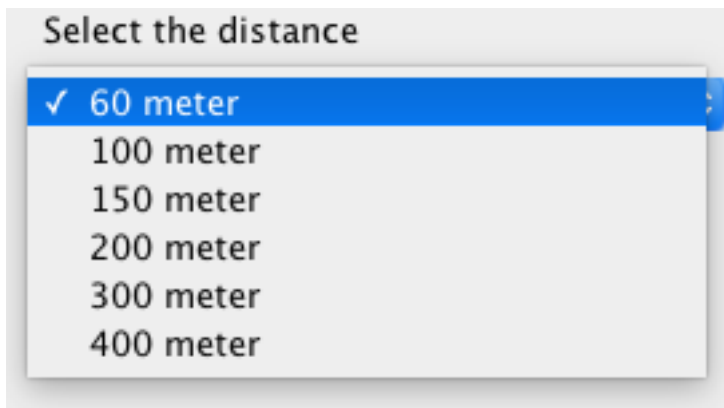


Figure 14: Graphical visualization of combo-box from figure 13

```

JComboBox comboBox = new JComboBox();
comboBox.setBounds(36, 72, 229, 20);
try {
    Scanner scan = new Scanner(new File("per.csv"));
    while(scan.hasNextLine()){
        String nam = scan.nextLine();
        String name = nam.split(",")[0];
        String gen = nam.split(",")[1];
        if(gen.contains("Men")){
            comboBox.addItem(name);
        }
    }
    scan.close();
} catch (FileNotFoundException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
} catch (IndexOutOfBoundsException e ){}

```

Figure 15: The code for retrieving name from per.csv file and putting it into the combobox

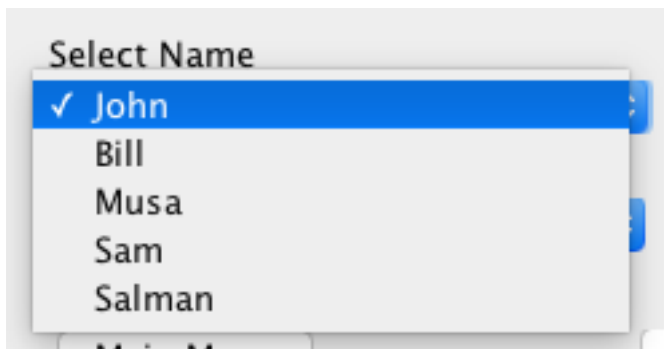


Figure 16: Graphical visualization of combo-box from figure 15

3. Creating the tables to match up .csv text format: Tables were created for the output to match the tabled .csv file format.

```

JLabel jb5 = new JLabel("400 meter");
jb5.setHorizontalAlignment(0);
jb5.setVerticalAlignment(0);
jb5.setBorder(BorderFactory.createLineBorder(Color.black));
JLabel jbh5 = new JLabel("400 meter");
jbh5.setHorizontalAlignment(0);
jbh5.setVerticalAlignment(0);
jbh5.setBorder(BorderFactory.createLineBorder(Color.black));
panel_2.add(jb5);
panel_3.add(jbh5);

```

Figure 17: Code for the 400m boxes in output Code.



Figure 18: Graphical visualization of distance box from code in figure 17

File Handling:

When Handling many variables; age, names, dates and times, I felt it was best to not use a .TXT file as they are less complex and lack the space needed to display all the elements above effectively. Thus, I was prompted to create unique .CSV files that automatically save in the form of a table to hold all values.

```
try {File ft60 = new File("date100 meter.csv");
Scanner scan = new Scanner(ft60);
String news1 = "";
while(scan.hasNextLine()){
    String news = scan.nextLine();
    int sp = news.split(",").length;
    if(sp>num){
        String news2 = "";
        String[] flow = news.split(",");
        for(int i=0;i<flow.length;i++){
            if(i==num){
                continue;
            }
            if(i==0){
                news2+=flow[i];
            }else{
                news2+=","+flow[i];
            }
        }

        news1+=news2+"\n";

    }else{
        news1+=news+"\n";
    }
}

File ftn= new File("date100 meter.csv");
FileWriter fw = new FileWriter(ftn);
fw.write(news1);
fw.close();
} catch (FileNotFoundException e) {

} catch (IOException e) {

}
```

Figure 19: Example of one of many .csv files created for event


```
File f= new File("per.csv");  
File fnew = new File(textField+".csv");  
try {  
    FileWriter fw = new FileWriter(f,true);  
    String name = textField.getText();  
    fw.write(name+","+gender+",true\n");  
    fw.close();  
}
```

Figure 20: Example of per.csv file used to create new file to save all the athletes.

Error Handling:

Error handling is a very important facet of this product as it was errors that promoted me to create it in the first place. In terms of errors within the program, I have made sure to set up several `FileNotFoundException`s and `IOException`s. Furthermore, there are also several `try` and `catch` statements within the code.

```
for(int i=0;i<jt.size();i++){
    fin+=" "+jt.get(i).getText() ;
    try{
        double db = Double.parseDouble(jt.get(i).getText());
    }catch(Exception ed){
        JOptionPane.showMessageDialog(null, "Please enter the correct Value");
        fw.close();
        return ;
    }
}
```

Figure 10: Code containing a try-catch statement

Figure 10 is algorithm made to illustrate one of the examples of error handling in the program. This is when the user is prompted to enter the times for the athletes. If user does not enter a Double or Int value, the error shall appear in form of a dialogue box prompting them to re-enter their data in the correct format.

```
File ftn= new File("date100 meter.csv");
FileWriter fw = new FileWriter(ftn);
fw.write(news1);
fw.close();
} catch (FileNotFoundException e) {
```

Figure 1: Code containing a try-catch statement that will catch if file for 100m time is not present.