Criterion C: Development

List of Techniques used:

1. Inheritance
2. Random Access File
3. Data Input
4. Files Writing Methods
5. One dimensional array
6. Text field focus lost

Inheritance:

Super class

```
public class Employee {
    private String name;
    private int age;
    private String gender;
    private double salary;
    private String phoneNum;
    private String address;
    private String department;
    private int employeeNumber;
    private boolean isManager;
    private int vacation;
    public Employee(){// constructor:Set initial values for class
        name = " ";
        age = 0;
        salary=0.00;
        phoneNum=" ";
        address=" ";
        department=" ";
        gender=" ";
        employeeNumber=0;
        isManager=false;
        vacation = 15;
    }
```

```
    public Employee(String name, int age, String sex, double sal,String cell,String home,
            String title, int ID,int years, boolean manager){// overload constructor
        this.name = name;
        this.age = age;
        gender=sex;
        salary=sal;
        phoneNum=cell;
        address=home;
        department=title;
        employeeNumber=ID;
        isManager=manager;
    }
public void loadinfo(String name, int age,String sex,double sal,String cell,String home,
            String title, int ID,boolean manager){//mutator
        this.name = name;
        this.age = age;
        salary=sal;
        gender=sex;
        phoneNum=cell;
        address=home;
        department=title;
        employeeNumber=ID;
        isManager=manager;
}
```

Extended class:

```
 */
public class Managers  extends Employee {
    private int code;

    public Managers(){
        super();
        code=000000;
    }
    public Managers(String name, int age, String sex, double sal,String cell,String home,
            String title, int ID,int years,boolean manager ,int password){// overload constructor
        super(name,age,sex,sal,cell,home,title,ID,years,manager);
        this.code = password;
    }

    public void changeCode(int temp){
        code = temp;
    }
    public int getCode(){
        return code;
    }
}
```

```
    gender = jComboBox2.getSelectedItem().toString();
    temp.loadinfo(fname + lname, Age, gender, salary, phoneNum, Address, department, employeeNumber, manager);
    if (jComboBox3.getSelectedItem().toString().equalsIgnoreCase("Manager")) {
        manager = true;
        temp.ifManager(manager);
        input.loadinfo(fname + lname, Age, gender, salary, phoneNum, Address, department, employeeNumber, manage
        getEntered(input);
        input.changeCode(Integer.parseInt(JOptionPane.showInputDialog("Please set a code for authorizing absence
        int index=posn.getSelectedIndex();
        if(checkUnique(index,raf2,51)){
            writeInfo(temp, raf);
            writeInfo(input, raf2);
        }
        else{
            JOptionPane.showMessageDialog(null, "Not able to create manager under"
                    + " selected department \nBecause department manager exist!", "Conflict", 1);
        }
    }
```

Every instance of the Employee class will hold the data of an employee while its extended class, Manager, holds one additional integer property. The program will detect the item selected for ComboBox3 assigned for position in the department and create a corresponding instance of class object and will create an additional Manager instance if the item selected is Manager.

Inheritance is integrated in programing because it applies closely to the situation. In my client's company, managers and employees have a lot of information in common. But managers have an additional piece called authorization code which is used to grant employees in his/her department absences of occupation, having a separate extended class to hold information of a manager allows users to have a clear view of the human resource in the company.

Source:

Thakur, D. Multilevel Inheritance. Retrieved April 4, 2018, from http://ecomputernotes.com/java/inheritance/multilevel-inheritance-in-java

Random Access File:

Declaring File Instance:

```
RandomAccessFile raf;  // define first file instance
RandomAccessFile raf2; // second
String FileName = ("EmployeesFile.txt");
File inFile = new File(FileName);
String File2 = ("ManagersFile.txt");
File sndFile = new File(File2);
boolean restart;
public AddStaff() {
    initComponents();
    restart=false;
    //Checking file state  DO NOT MODIFY!!!!!!!!!!!!!!!!!!!!!!!!!
    if (inFile.exists() && inFile.length() != 0) {
        try {
            raf = new RandomAccessFile(inFile, "rw"); // open file
            System.out.println("File opened");
        } catch (IOException ex) {
            ex.printStackTrace();
        }

    } else {
        JOptionPane.showMessageDialog(null, "Employee profile could not be found, creating a blank file");
        String ten = "          ";
        String fields[] = new String[250];
        for (int i = 0; i < fields.length; i++) {
            fields[i] = ten + ten + ten + ten + ten + ten + ten + ten + ten + ten + ten + " ";//111 spaces
        }
        writefile(fields, FileName);
        restart=true;
    }
    // Employees File opened
```

With any interface opened, the program with attempt to open the files that were created before, or if any file is not spotted, it will create blank files and initialize them accordingly. For EmployeesFile, the program will initialize 250 lines of record with equal length of spaces, which was calculated to be long enough to hold all necessary personal details of an employee or a manager. Same with ManagersFile.

RAF access method:

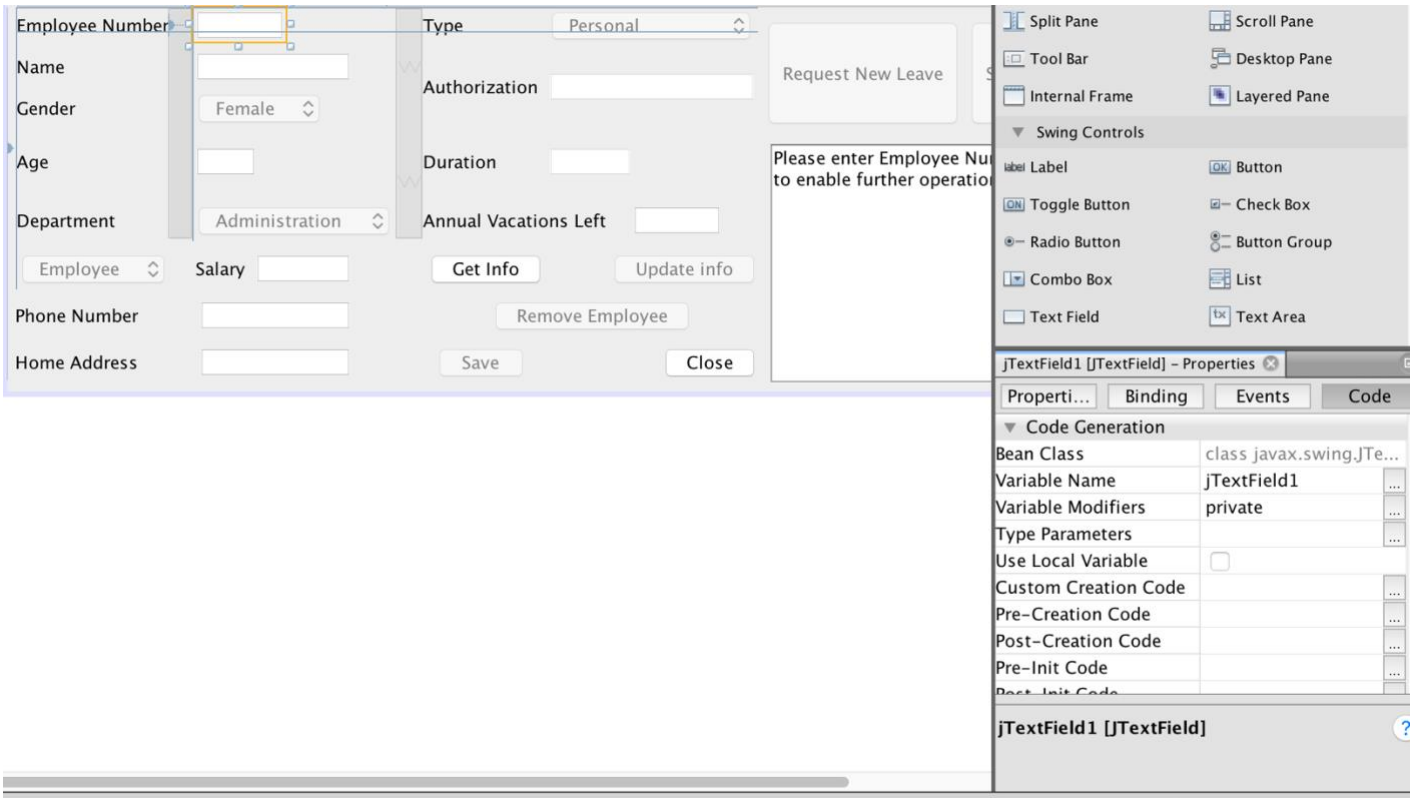
```
public void loadEmployee(String in) {
    einput.changeAge(Integer.parseInt(noSpace(in.substring(20, 22))));
    einput.changeAddress(noSpace(in.substring(50, 70)));
    einput.changePhone(noSpace(in.substring(35, 47)));
    einput.updateVacation(Integer.parseInt(noSpace(in.substring(47, 50))));
    einput.changePosn(noSpace(in.substring(70, 85)));
    einput.changeSal(Double.parseDouble(noSpace(in.substring(25, 35))));
    einput.changeName(noSpace(in.substring(0, 20)));
    einput.changeENum(Integer.parseInt(noSpace(in.substring(85, 90))));
    einput.changeGender(noSpace(in.substring(90, 100)));
    if (noSpace(in.substring(100, 111)).equalsIgnoreCase("Manager")) {
        einput.ifManager(true);
    } else {
        einput.ifManager(false);
    }
    //System.out.println(einput.getAll());
}
```

Random Access File is the method chosen for data storage and access. In this very project, data pulling method must have the capacity to output the record as needed. Instead of outputting the whole file. It must also be capable of being written at any given address. Compare to using a Java Database, using random access file is better and more convenient in terms of record keeping.

Source:

Thakur, D. Random Access File. Retrieved April 4, 2018, from http://ecomputernotes.com/java/stream/randomaccessfile

File Access:




```
final int length = 112; // 111 characters plus EOFL
if (jTextField1.getText().isEmpty()) {
    JOptionPane.showMessageDialog(null, "Please enter Employee Number"
            + " to enable further operations");

} else {
    index = Integer.parseInt(jTextField1.getText()) - 1;
    try {
        raf.seek(length * index);
        in = raf.readLine();
        for (int i = 0; i < in.length(); i++) {
            if (chars.indexOf(in.charAt(i)) >= 0) {
                checked = true;
                break;
            } else {
                JOptionPane.showMessageDialog(null, "No record under entered employee number!");
                break;
            }
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

If a valid employee number, x, is enter, the program will fetch the x^{th} record in the EmployeesFile and put the whole line into a String variable for later processes. Otherwise, there will be an error message displayed through JOptionPane message dialogue. The same technique is used to access ManagersFile when an authorization code is required for approving an absence request. It is used again to get all records of the desired employee and process it in the Individual Summary interface.

Having different files that can be identified by the same primary key enables connections between data storages. The idea of information hiding and arrangement is represented.

Source:

Thakur, D. Random Access File. Retrieved April 4, 2018, from http://ecomputernotes.com/java/stream/randomaccessfile

File Writing Methods:

Sequential File Writer:

```
public static void writefile(String variables[], String fName) {
    try {
        PrintWriter out = new PrintWriter(new FileWriter(fName));
        for (int i = 0; i < variables.length; i++) {
            out.println(variables[i]);  // output the values to the "out" stream
        }
        out.close();  // output to file
    } catch (java.io.IOException e) {
        System.out.println(e);
    }
}
```

Sequential File Writer is used when writing blank file with equal length of space for records. Therefore an initialized file can be created with least amount of code.

Source:

Thakur, D. FileReader and FileWriter. Retrieved April 4, 2018, from http://ecomputernotes.com/java/stream/filereaderwriter

RAF writing method:

```
public void writeInfo(Employee temp, RandomAccessFile raf) {
    final int length = 112; // 111 characters plus EOFL
    int count = temp.getENUM() - 1;
    try {
        raf.seek(count * length);
        raf.writeBytes(temp.getName());
        raf.seek(count * length + 20);
        raf.writeBytes(Integer.toString(temp.getAge()));
        raf.seek(count * length + 25);
        raf.writeBytes(Double.toString(temp.getSal()));
        raf.seek(count * length + 35);
        raf.writeBytes(temp.getPhone());
        raf.seek(count * length + 47);
        raf.writeBytes(Integer.toString(temp.GetDays()));
        raf.seek(count * length + 50);
        raf.writeBytes(temp.getAddress());
        raf.seek(count * length + 70);
        raf.writeBytes(temp.getPosn());
        raf.seek(count * length + 85);
        raf.writeBytes(Integer.toString(temp.getENUM()));
        raf.seek(count * length + 90);
        raf.writeBytes(temp.getGender());
        raf.seek(count * length + 100);
        raf.writeBytes(temp.getIfManager());
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Once data is entered and processed correctly, a series of raf.writeBytes will output data to the desired position as schemed. The use of sub-method reduces repetition of coding, which shortens the time it takes to complete the program. The idea of encapsulation and isolating errors should also be highlighted.

Initializing Record of Absence File:

```
        } else {// exception when file is not found
            JOptionPane.showMessageDialog(null, "Record file could not be found, creating a blank file");
            String ten = "          ";
            String fields[] = new String[1500];
            for (int i = 0; i < fields.length; i++) {
                fields[i] = ten + ten + ten + ten + ten + ten + ten;//70 spaces
            }
            writefile(fields, File3);
            JOptionPane.showMessageDialog(null, "Blank file created");
            //create a file if not created before for first data entrance.
            // write in default values
            try {
                raf3 = new RandomAccessFile(record, "rw"); // open file
                System.out.println("File opened");
                for (int i = 0; i < 1200; i += 6) {
                    raf3.seek(i * recL + 20);
                    // raf3.writeBytes(Integer.toString(c));//Corresponding eNUM
                    raf3.seek(i * recL + 25);
                    raf3.writeBytes("Personal");//Corresponding Type
                    raf3.seek(i * recL + 45);
                    raf3.writeBytes("0.00");//default time in days
                }
                //
                for (int j = 1; j < 1201; j += 6) {
                    raf3.seek(j * recL + 20);
                    //raf3.writeBytes(Integer.toString(c));//Corresponding eNUM
                    raf3.seek(j * recL + 25);
                    raf3.writeBytes("Business");//Corresponding Type
                    raf3.seek(j * recL + 45);
                    raf3.writeBytes("0.00");//default time in days
                }
                //
                for (int h = 2; h < 1202; h += 6) {
                    raf3.seek(h * recL + 20);
                    //raf3.writeBytes(Integer.toString(c));//Corresponding eNUM
                    raf3.seek(h * recL + 25);
                    raf3.writeBytes("Marriage");//Corresponding Type
                    raf3.seek(h * recL + 45);
                    raf3.writeBytes("0.00");//default time in days
                }
                //
                for (int k = 3; k < 1203; k += 6) {
                    raf3.seek(k * recL + 20);
                    //raf3.writeBytes(Integer.toString(c));//Corresponding eNUM
                    raf3.seek(k * recL + 25);
                    raf3.writeBytes("Pregnancy");//Corresponding Type
                    raf3.seek(k * recL + 45);
                    raf3.writeBytes("0.00");//default time in days
                }
                //
                for (int l = 4; l < 1204; l += 6) {
                    raf3.seek(l * recL + 20);
                    //raf3.writeBytes(Integer.toString(c));//Corresponding eNUM
                    raf3.seek(l * recL + 25);
                    raf3.writeBytes("Annual Vacation");//Corresponding Type
                    raf3.seek(l * recL + 45);
                    raf3.writeBytes("0.00");//default time in days
                }
                //
                for (int m = 5; m < 1205; m += 6) {
                    raf3.seek(m * recL + 20);
                    // raf3.writeBytes(Integer.toString(c));//Corresponding eNUM
                    raf3.seek(m * recL + 25);
                    raf3.writeBytes("Sickness");//Corresponding Type
                    raf3.seek(m * recL + 45);
                    raf3.writeBytes("0.00");//default time in days
```

Record of Absence file will be set to default values by the method above, which is for every employee, there will be six lines of records with 0.00 days of absence in place.

It is easier to keep track of the total length of absence in days regardless of what type it is. The program will automatically update the double value whenever the user inputs a new data entry. If an employee is on duty every working day, he/she should receive all disposable income after taxation and adjustment due to insurance paid by the company.

Record of Absence Update method:

```
if (record.exists() && record.length() != 0) {
    try {
        raf3 = new RandomAccessFile(record, "rw"); // open file
        System.out.println("File opened");
        // all good, write new instance to record
         System.out.println("Type- "+leave.getType());
        if (leave.getType().equalsIgnoreCase("Personal")) {
            count = 0;
        } else if (leave.getType().equalsIgnoreCase("Business")) {
            count = 1;
        } else if (leave.getType().equalsIgnoreCase("Marriage")) {
            count = 2;
        } else if (leave.getType().equalsIgnoreCase("Pregnancy")) {
            count = 3;
        } else if (leave.getType().equalsIgnoreCase("Annual Vacation")) {
            count = 4;
            int change=einput.GetDays();
            change-=leave.getLength();
            raf.seek(112 * (leave.getId()-1) + 47);
            raf.writeBytes("   ");//wipes the original record
            if(change<10&&change>=0){
                raf.seek(112 * (leave.getId()-1) + 48);
                raf.writeBytes(Integer.toString(change));
            }
            else if(change>=10&&change<15){
                raf.seek(112 * (leave.getId()-1) + 47);
                raf.writeBytes(Integer.toString(change));
            }

        } else {
            count = 5;
        }
        System.out.println("Index is "+count);
```

```
double sum;
String input;
//writing in
raf3.seek(((leave.getId() - 1)*6 + count) * recL);
//System.out.println("Fetching "+(leave.getId()-1+count)*recL);
raf3.writeBytes(leave.getName());
raf3.seek(((leave.getId() - 1)*6 + count) * recL + 20);
raf3.writeBytes(Integer.toString(leave.getId()));
raf3.seek(((leave.getId() - 1)*6 + count) * recL + 45);
input = noSpace(raf3.readLine()).substring(0, 4);
System.out.println("Reads" + input);
sum = Double.parseDouble(input);
sum += leave.getLength();
System.out.println("Total time changed to " + sum);
raf3.seek(((leave.getId() - 1)*6 + count) * recL + 45);
raf3.writeBytes(Double.toString(sum));
raf3.seek(((leave.getId() - 1)*6 + count) * recL + 50);
raf3.writeBytes(leave.getDepartment());

// new data entrance written
```

After an absence request is authorized, the program will first match the type selected by the user with the line of record assigned to the select employee's employee number. I used an integer counter and a combination of if statements to convert String values into an instruction setting the pointer to the intended position in file so that data can be updated to the file accurately with raf.writeByte statement with a complex position parameter.

One dimensional array:

```
String input []=new String[6];
input=loadRec(einput,Personal,Business,Marriage,Pregnancy,Vacation,Sickness);
double sum=0;
for(int i=0;i<6;i++){
   sum+=Double.parseDouble(input[i]);
}
t1.setText(input[0]);
t2.setText(input[1]);
t3.setText(input[2]);
t4.setText(input[3]);
t5.setText(input[4]);
t6.setText(input[5]);
```

```
public String[] loadRec(Employee temp,String p,String b, String m,String preg,String v,String s){
    int count = temp.getENUM()-1;
    final int recLength = 71;

    try{
            raf3.seek((count*6)*recLength);
            p=raf3.readLine().substring(45, 49);
            b=raf3.readLine().substring(45, 49);
            m=raf3.readLine().substring(45, 49);
            preg=raf3.readLine().substring(45, 49);
            v=raf3.readLine().substring(45, 49);
            s=raf3.readLine().substring(45, 49);
            System.out.println("Records are\n"+ p+"\n"+b+"\n"+m+"\n"+preg+"\n"+v+"\n"+s);
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
```

A one dimensional array holds all the absence records relevant to the employee reached when user click on search in Individual Summary interface. Once that is done, the loadRec method will process the array and returns another array with absence duration of all types in the format of Strings.

Source:

Thakur, D. One Dimensional Array. Retrieved April 4, 2018, from http://ecomputernotes.com/java/array/one-dimensional-array

Text Field Focus Lost:

```
private void eNumFocusLost(java.awt.event.FocusEvent evt) {
    // TODO add your handling code here:
    String in = (eNum.getText());
    String ints = "1234567890";
    for (int i = 0; i < in.length(); i++) {
        if (ints.indexOf(in.charAt(i)) < 0) {
            JOptionPane.showMessageDialog(null, "Field not enterred correctly", "Warning", 1);
            break;
        }
        if(Integer.parseInt(eNum.getText())>0&&Integer.parseInt(eNum.getText())<=200){
        if(checkUnique(Integer.parseInt(eNum.getText())-1,raf,112)){
            System.out.println("Unique primary Key!");
        }
        else{
            JOptionPane.showMessageDialog(null, "Employee Number Has been Taken!", "Warning", 1);
            eNum.setText("");
        }
        }
        else{
            JOptionPane.showMessageDialog(null, "Please enter a number from 1 to 200", "Warning", 1);
            eNum.setText("");
        }
    }
```

Focus Lost event of the applied text field enables a mandatory comparison between the entered information and desired information. In this particular case, it will check if the employee number has been taken. If so, it will display an error message and clear the text box.

The use of this technique makes sure that inputs entered by the user are valid and ready to be processed. Thus, possible errors, failure of functioning could be eliminated.

Word Count: 795