

Criterion C- Development

Introduction

The IDE, Netbeans, was used to create a Java program that displays a table of all of the tasks that an individual wants to complete: an organized to-do list. Using Java's Swing tools, the graphical user interface allows the program to sort tasks, showcase progress, and keep track of what is completed.

Word Count: 52

List of Techniques

- For loops
- Sorting (bubble sort)
- Use of flag values ("not set yet", 999)
- Converting (parsing)
- GUI tabs/ different panels
- Encapsulation & Inheritance
- Simple and compound selection (if statements)
- Method returning a value
- Arrays
- User defined objects from "template" class
- Error handling

Parsing example

```
Integer.parseInt(timeToSpendOnTaskTF.getText()),  
Integer.parseInt((String)difficultyLevelComboBox.getSelectedItem()), false, 0);
```

Flag values examples

```
private String name = "not set yet";  
private String description = "";  
private int time = 999;  
private int difficulty = 999;  
private boolean completed = false;  
private int actualtime = 999;
```

Using bubble sort

```
public class SortingAndSearching {  
    public void sortByDifficulty(Task [] scheduleArray) {  
        int n = scheduleArray.length;  
        boolean sorted = false;  
        while (!sorted) {  
            n--;  
            sorted = true;  
            for (int i=0; i < n; i++) {  
                if (scheduleArray[i].getDifficulty() > scheduleArray[i+1].getDifficulty()) {  
                    Task temp = scheduleArray[i];  
                    scheduleArray[i] = scheduleArray[i+1];  
                    scheduleArray[i+1] = temp;  
                    sorted = false;  
                }  
            }  
        }  
    }  
}
```

Structure of Program

What:

The central feature of the program is focused on the GUI class. The “template” Task class instantiates 6 different attributes/ objects and returns a reference with the allocated memory in the GUI. Another class called SortingAndSearching contains bubble sorting algorithms in order to sort different tasks based on specifications that are called in the main GUI.

Why:

The program is based on OOP. Encapsulation occurs within the “Task” template class to enable manipulation of attributes and the creation of multiple objects from the same class (see figure 1). This makes it easier to understand and maintain. Inheritance is seen within the GUI class because it is extending to the Task class, thus it creates reusability (because of public methods), and data hiding (certain data is private thus cannot be altered from derived class). The GUI class contains the parts of the program that enables the Java Swing Tools in the design to function (see figure 2). Overall, OOP makes it easier to develop the program by allowing control over the state of the data more efficiently. Along with having a separate class for bubble sorting, it provides organization and an easier way to visualize the code.

Word Count: 195

Figure 1: Main GUI using two other classes.

```
public class prototypeMainGUI extends javax.swing.JFrame {  
    private Task[] tasks = new Task[10];  
    private int counter = 0;  
    private SortingAndSearching sortAndSearch;  
  
    public prototypeMainGUI() { ...4 lines }  
  
    public void myIntComponents() { ...9 lines }  
  
    private void refreshDisplay() { ...13 lines }
```

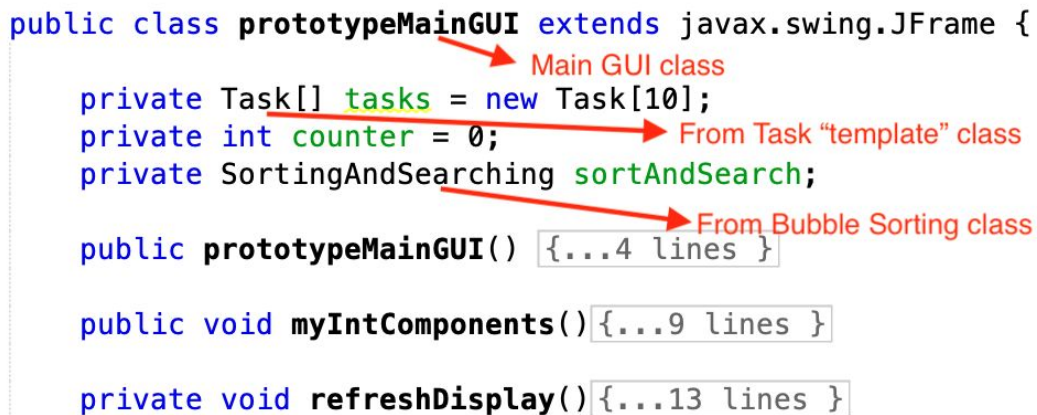


Figure 2: Use of Java Swing Tools in the GUI.

```
private javax.swing.JMenuItem aboutMenuItem;  
private javax.swing.JToggleButton completedTaskButton;  
private javax.swing.JLabel completedTaskLabel;  
private javax.swing.JTextField completedTaskTF;  
private javax.swing.JMenuItem contentsMenuItem;  
private javax.swing.JMenuItem copyMenuItem;  
private javax.swing.JMenuItem cutMenuItem;  
private javax.swing.JMenuItem deleteMenuItem;  
private javax.swing.JTextArea descriptionArea;  
private javax.swing.JComboBox<String> difficultyLevelComboBox;  
private javax.swing.JLabel difficultyLevelLabel;  
private javax.swing.JTable displayTable;  
private javax.swing.JMenu editMenu;  
private javax.swing.JMenuItem exitMenuItem;
```

Data Structures Used

An Array of Task objects was used containing 6 attributes in the Task template class such as name, description, time, etc. This allows multiple values to be stored in one variable, thus each slot holds a different object which makes it easier considering there is less copying and pasting and can be reused for convenience.

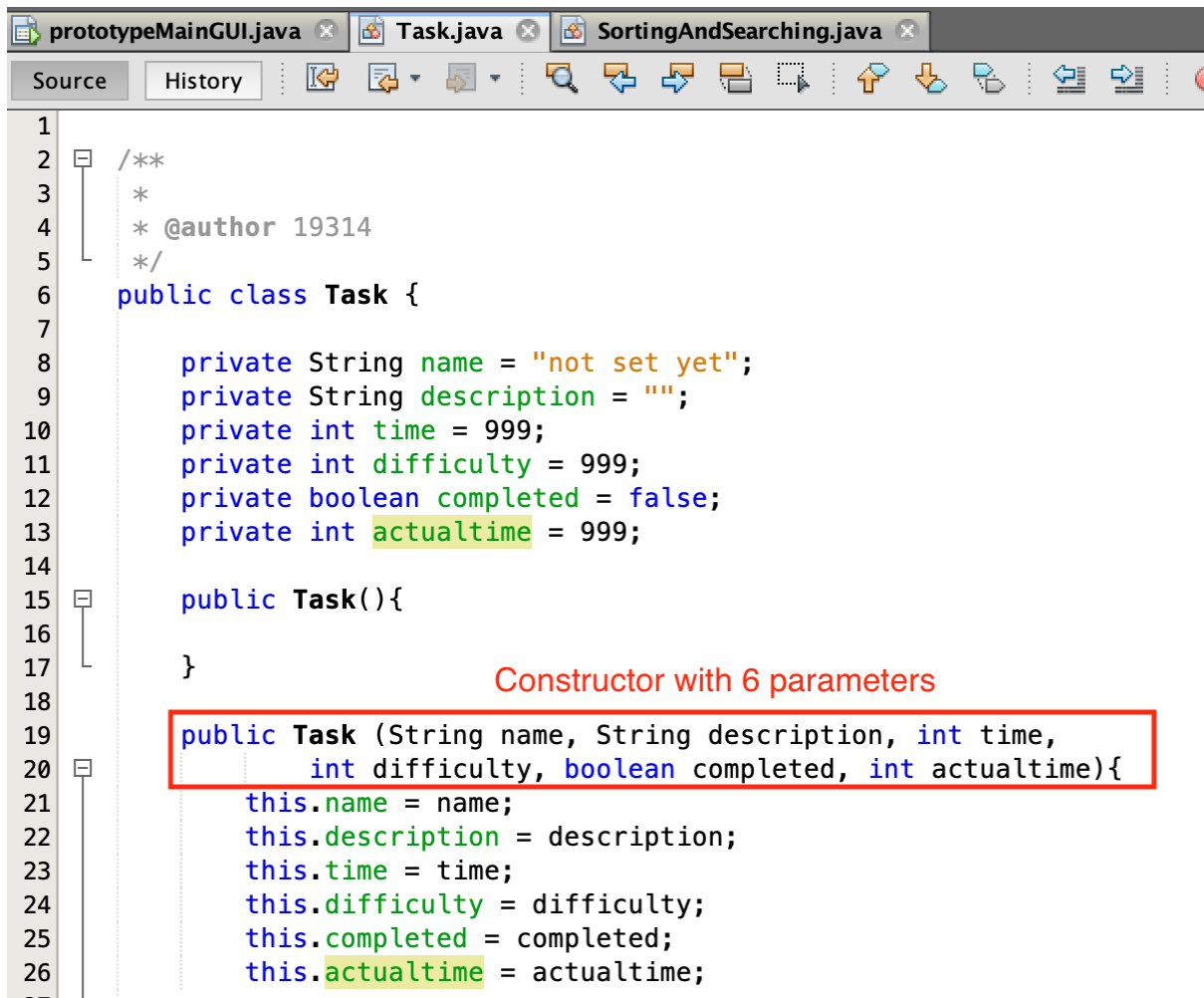
Word Count: 55

Figure 3:

```
public class prototypeMainGUI extends javax.swing.JFrame {  
    private Task[] tasks = new Task[10];  
    private int counter = 0;  
    private SortingAndSearching sortAndSearch;
```

Array from Task class

Figure 4:



```
1  
2 /**  
3  *  
4  * @author 19314  
5  */  
6 public class Task {  
7  
8     private String name = "not set yet";  
9     private String description = "";  
10    private int time = 999;  
11    private int difficulty = 999;  
12    private boolean completed = false;  
13    private int actualtime = 999;  
14  
15    public Task(){  
16  
17    }  
18  
19    public Task (String name, String description, int time,  
20                int difficulty, boolean completed, int actualtime){  
21        this.name = name;  
22        this.description = description;  
23        this.time = time;  
24        this.difficulty = difficulty;  
25        this.completed = completed;  
26        this.actualtime = actualtime;
```

Constructor with 6 parameters

Main Unique Algorithms

A progress bar from Java Swings Tools was used to capture the client's completion of tasks. It calculates how many tasks they have completed out of the total number of tasks they input. Basic math was used: division (as seen in the figure below).

```
private void progressBarButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    double numberCompleted = 0;  
    for(int i = 0; i < counter; i++){  
        if(tasks[i].getCompleted()==true){  
            numberCompleted++;  
        }  
    }  
    double percentCompleted = (numberCompleted/counter) * 100;  
    progressBar.setValue((int)percentCompleted);  
}
```



- A “for loop” loops through all of the inputted tasks using the counter variable.
- The “if statement suggests that if getCompleted == true, this means that a task is completed and will follow through with the rest of the code to become a part of the “numberCompleted” value.
- If the getCompleted is true, it will calculate the percentage using the formula (numberCompleted divided by the total number of tasks, which is set as a counter) multiplied by 100 to get the percentage.
- To display the percentage on the progress bar, setValue was used.

This specific code is especially efficient because it reuses an object from the Task class and does simple math to calculate and display all of the completed tasks. This is very useful for the client because Mr. Eng is a visual person and would like to see his progress, this can also motivate him to complete more tasks to reach 100%.

Word Count: 197

Algorithm	Purpose
Input task data	Read text-file (created by client)
Display a To-Do list	Easy-to-read visual for the client
Sort tasks in the To-Do list	The client can sort the tasks by difficulty and time
Input updated data	A new table displays which tasks have been completed and how long it took

Pseudocode for Inputing task data:

Create = new task (take in "gets" for each attribute) (Convert→ parse if needed)

tasks array total tasks using counter = task;

counter++

Set all attributes ("")

```
private void inputTaskButtonMouseReleased(java.awt.event.MouseEvent evt) {
    //when the client clicks on the button on the Input Task Tab, it will store the inputted values
    //Using parameters, the new "task" takes in the name, description, time, and difficulty level which
    //is set at false and 0
    Task task = new Task (taskNameTF.getText(), descriptionArea.getText(),
        Integer.parseInt(timeToSpendOnTaskTF.getText()),
        Integer.parseInt((String)difficultyLevelComboBox.getSelectedItem()), false, 0);

    tasks [counter] = task;
    counter++;

    //clear the text field
    taskNameTF.setText("");
    descriptionArea.setText("");
    timeToSpendOnTaskTF.setText("");
    difficultyLevelComboBox.setSelectedIndex(0);
    //text fields are cleared and ready for the next task to be inputted
}
```

Pseudocode for displaying To-Do list:

//create a method refreshDisplay

get row count

row = -1

For loop going through all tasks (counter)

next row

print "i" the variable

set variables for each object to each row

```

//method that will be used in To-Do List Display tab to update the table of tasks
private void refreshDisplay(){
    System.out.println("The length of the table is: " + displayTable.getRowCount());
    int nextRow = -1;
    for(int i = 0; i < counter; i++){
        nextRow++;
        System.out.println(i);
        displayTable.setValueAt(tasks[i].getName(),nextRow, 0);
        displayTable.setValueAt(tasks[i].getTime(),nextRow, 1);
        displayTable.setValueAt(tasks[i].getDescription(),nextRow, 2);
        displayTable.setValueAt(tasks[i].getDifficulty(),nextRow, 3);
    }
}

```

Pseudocode for inputting updated data:

String key = get all of the completed tasks

For loop through all tasks

Print key and get the name of the task with the task array [i]

If statement of key + getting the name of task [i]

setCompleted = true

Set actual time spent and parse/ convert it

Set the text fields ("")

```

private void completedTaskButtonMouseReleased(java.awt.event.MouseEvent evt) {
    /*Once the client has completed a task, they fill out which task has been completed
    and how long it actually took
    */
    String key = completedTaskTF.getText();

    for(int i = 0; i < counter; i++){
        System.out.println("key: " + key);
        System.out.println("tasks[counter].getName() " + tasks[i].getName());

        if(key.equalsIgnoreCase(tasks[i].getName())){
            tasks[i].setCompleted(true);
            tasks[i].setActualtime(Integer.parseInt(spentTimeOnTaskTF.getText()));
        }
    }
    //clear the text field box by ""
    completedTaskTF.setText("");
    spentTimeOnTaskTF.setText("");
}

```

Word Count: 172

Sample Input and Output of To-Do List:

Task name

Detail description of task

Time you would like to spend on the task (hours)

Level of Difficulty (1-10)

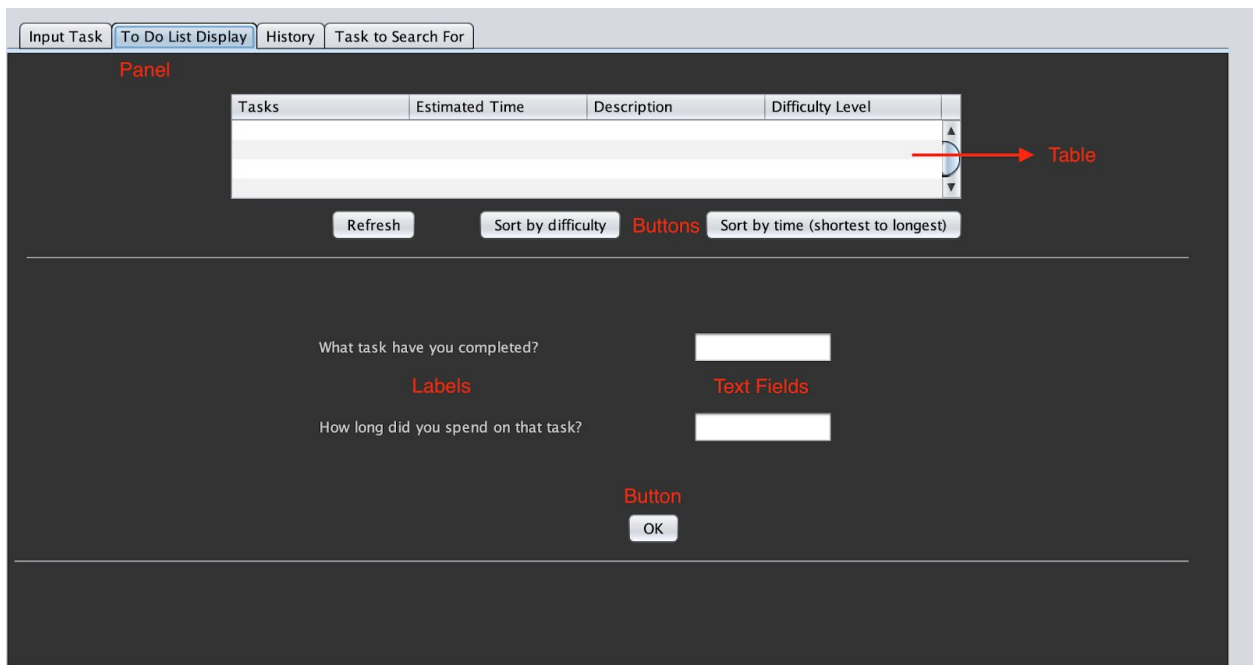
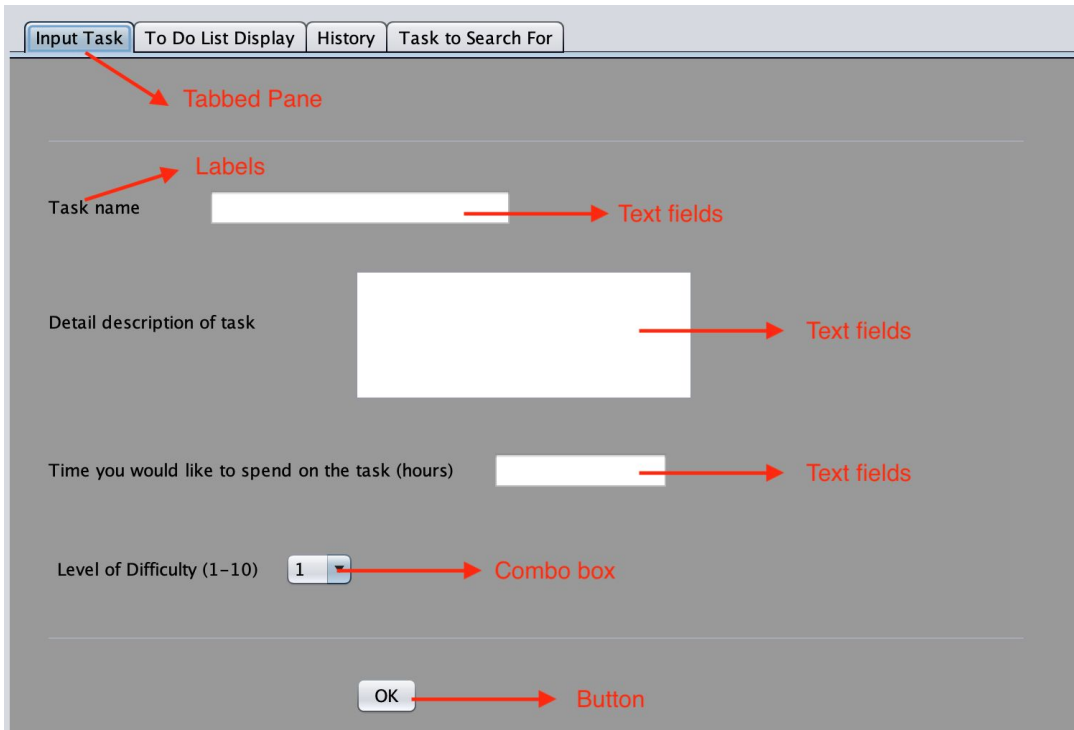
Tasks	Estimated Time	Description	Difficulty Level
Math homework	2	complete worksheet ...	7

User Interface/GUI Work

Some key features that were used from the Java Swings components can include:

- Text Fields
- Labels
- Buttons
- Tables
- Tabbed Pane
- Progress bar
- Combo Box
- Area Text Field

By using all of these features, it allows convenience for the development of the program and the client can easily understand through the simple “user-friendly” components. Text fields were often used to take in user input as well as buttons to allow the program to store the data in the Ram. Tables offer an easy way for the client to see all the displayed information. Along with tabbed panes, it is more organized to have the program in sections rather than having it all on one screen.



Input Task To Do List Display History Task to Search For

Label
Task to search for

Text field

Button
OK



Area Text Field

Software Tools Used

Netbeans is a popular Integrated Development Environment (IDE) platform for Java development. It can run on Windows, Linux, Solaris and macOS. In this case, Netbeans was the most appropriate IDE to use considering the development took place on a MacBook Air while working with Java. It is very convenient to use because it provides a wide range of tools and features which can boost productivity and efficiency. On Netbeans, users can add functionality and customize projects.

Word Count: 193

