

# 廈門大學



## 信息学院软件工程系

### 《JAVA 程序设计》实验报告

大作业 (在实验课已完成验收)

姓名：黄勛

学号：22920212204392

学院：信息学院

专业：软件工程

完成时间：2023.5.23

## 一、实验目的及要求

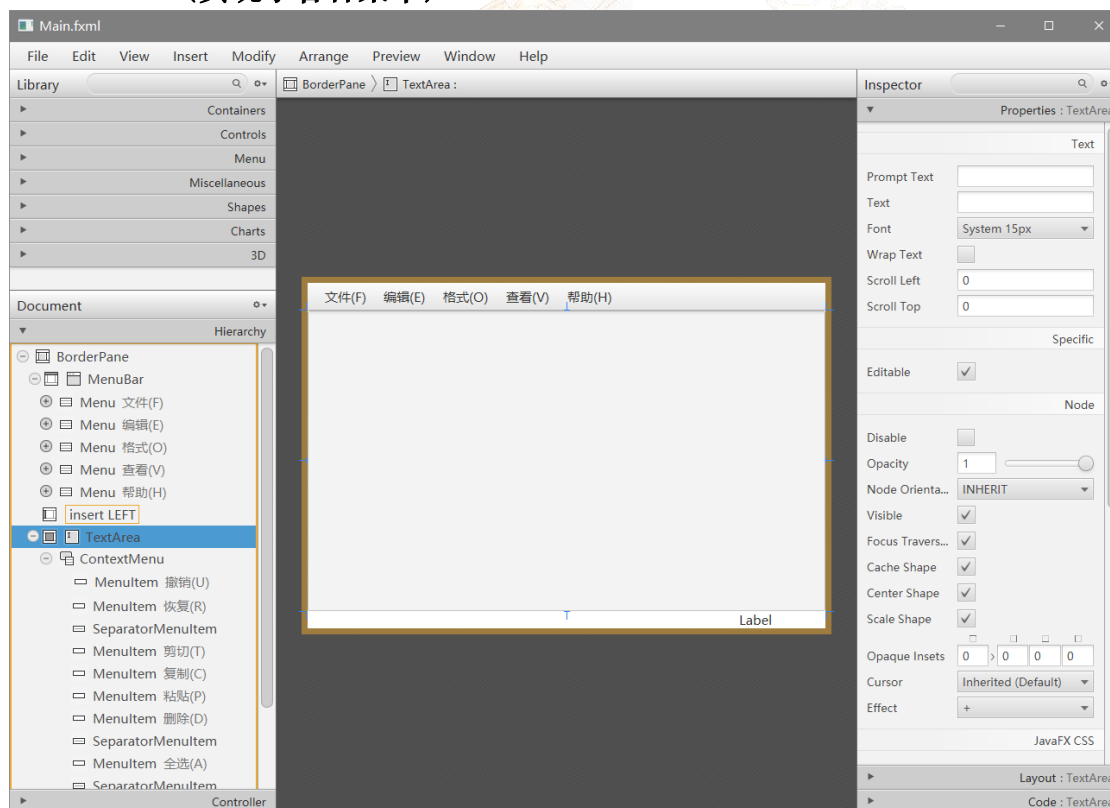
- 模拟 Windows 记事本，用 java 编程实现记事本的各项功能  
(在实验课已完成验收)

## 二、实现过程

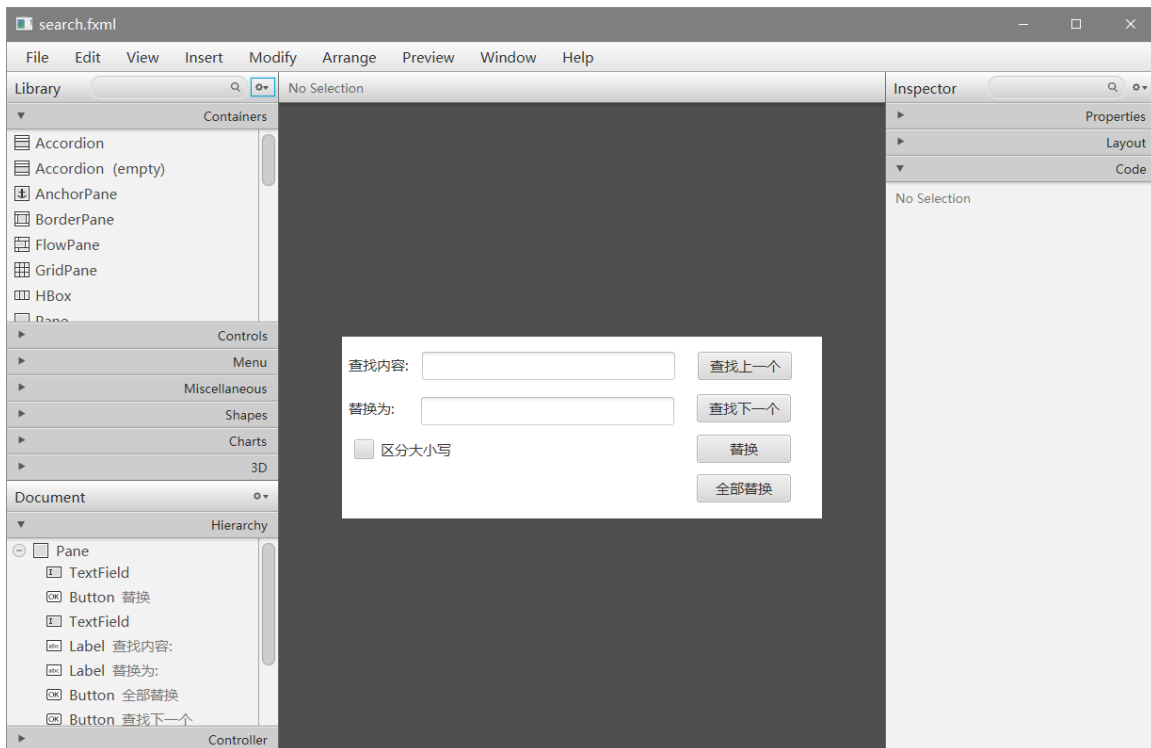
实验环境：Windows 10 21H2、jdk17、javafx scene builder、utf-8 编码

### (一) fxml 窗口设计：

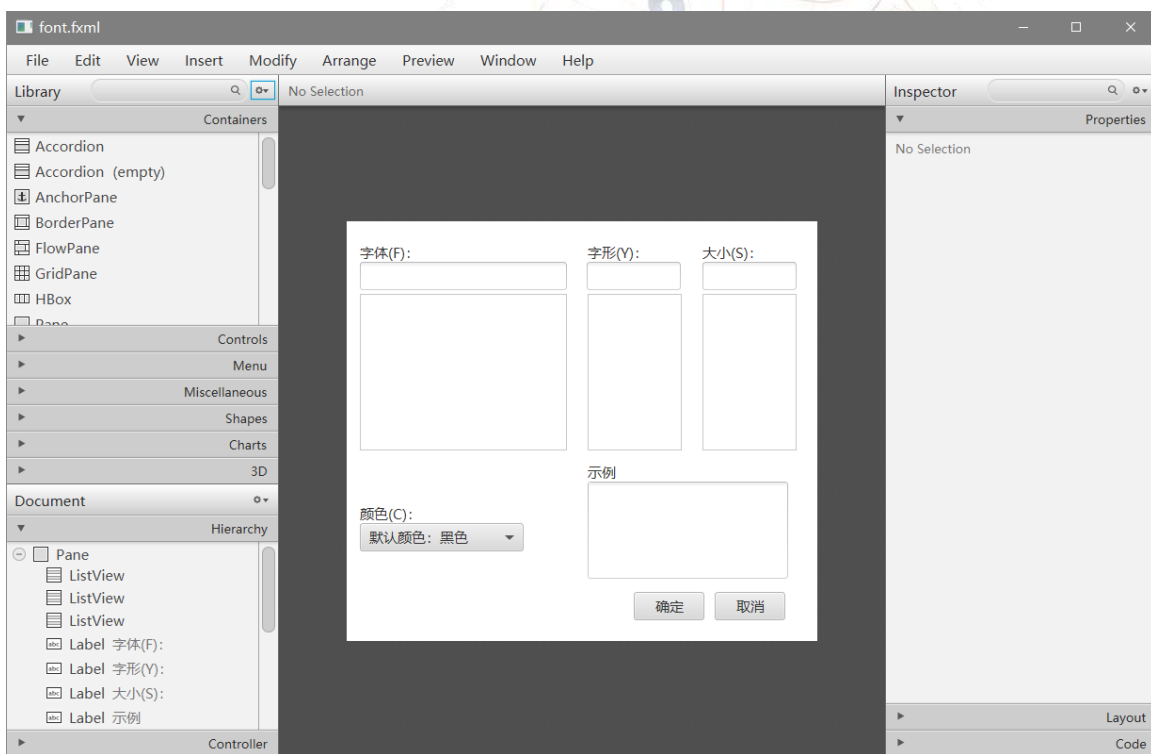
Main.fxml (实现了各种菜单)



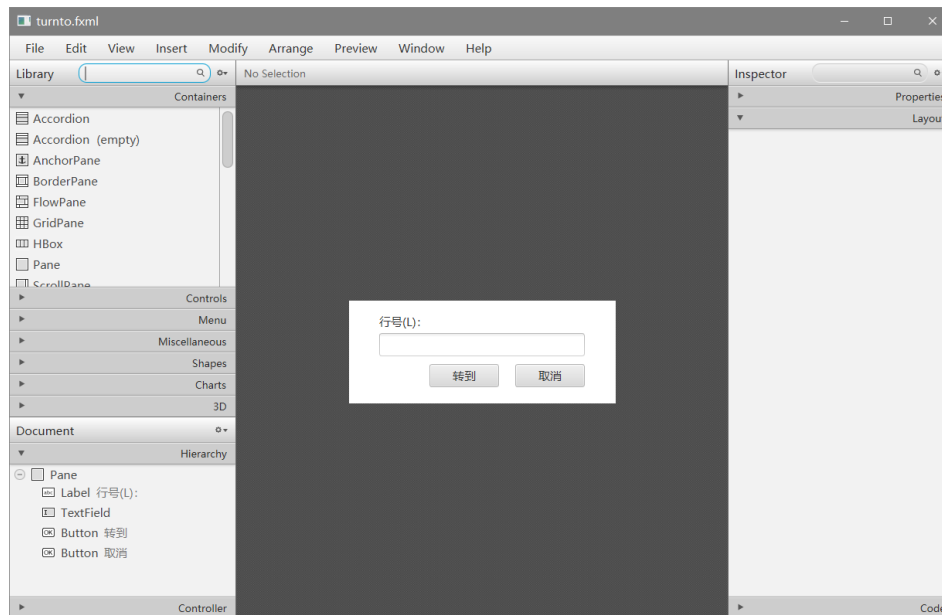
## search.fxml



## font.fxml

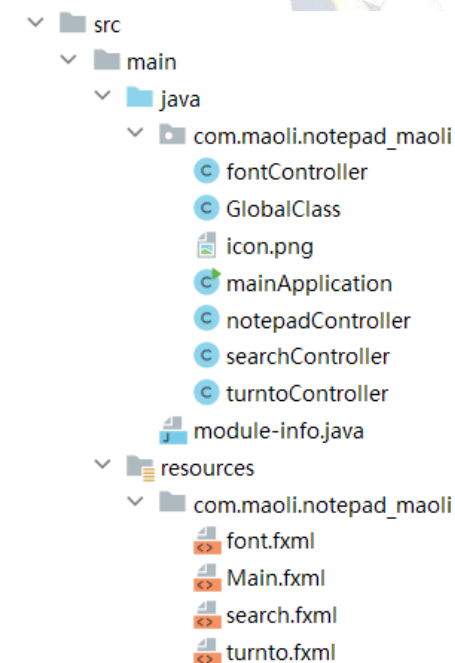


## turnto.fxml



## (二) 代码思路:

首先我设计了六个类，**mainApplication** 用于启动程序、**GlobalClass** 用于存储全局变量，以便在各个类中调用、**notepadController** 用于控制 notepad.fxml 的控制器，实现了记事本的大部分功能、**fontController** 用于字体选择的控制器，包括字体、字形、大小、颜色、**searchController** 用于搜索和替换窗口的控制器、**turntoController** 用于跳转到指定行窗口的控制器。



## mainApplication:

入口类，用于启动程序并加载主界面。下面是对代码的分析：

1. **mainApplication** 类继承自 **Application** 类，这是 JavaFX 应用程序的基类。
2. 在 **start** 方法中，创建了一个 **Stage** 对象作为主舞台，并设置了标题、场景和图标。

```
28  @Override public void start(Stage primaryStage) throws Exception
29  {
30      globals.put("global", global);
31      FXMLLoader fxmlLoader = new FXMLLoader();
32      fxmlLoader.setLocation(getClass().getResource("Main.fxml"));
33      fxmlLoader.setBuilderFactory(new JavaFXBuilderFactory());
34      Parent root = fxmlLoader.load();
35      primaryStage.setTitle("无标题 - 记事本");
36      primaryStage.setScene(new Scene(root));
37      primaryStage.getIcons().add(new Image("file:src/main/java/com/maoli/notepad_maoli/icon.png"));
38      primaryStage.show();
```

3. 使用 **FXMLLoader** 加载 **Main.fxml** 文档作为程序的主界面，并通过 **fxmlLoader.load()** 方法加载并获取根节点。
4. 获取主界面的控制器对象 **notepadController**，并将主舞台传递给控制器的 **mainStage** 方法。
5. 调用控制器的 **zoominit** 方法进行初始化操作。
6. 在主舞台关闭时，添加了一个关闭事件处理程序。如果当前编辑器状态不为 3（未保存状态），则会显示一个确认对话框，询问用户是否保存更改。根据用户的选择，执行相应的操作。

```
43  primaryStage.setOnCloseRequest(event -> {
44      if (controller.getStatus() != 3)
45      {
46          if (controller.getCurrentPath() == null)
47          {
48              Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
49              alert.setTitle("记事本");
50              alert.setHeaderText("你想将更改保存到 无标题 吗?");
51              Optional<ButtonType> result = alert.showAndWait();
52
53              if (result.get() == ButtonType.OK)
54              {
55                  if(controller.saveAs())
56                      primaryStage.close();
57              }else if (result.get() == ButtonType.CANCEL)
58              {
59                  primaryStage.close();
60              }
61          }
62          else
63          {
64              Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
65              alert.setTitle("记事本");
66              alert.setHeaderText("你想将更改保存到 " + controller.getCurrentPath() + " 吗?");
67              Optional<ButtonType> result = alert.showAndWait();
68              if (result.get() == ButtonType.OK)
69              {
70                  controller.save();
71                  primaryStage.close();
72              }
73              else if (result.get() == ButtonType.CANCEL)
74              {
75                  primaryStage.close();
76              }
77          }
78      }
79  });
```

## GlobalClass:

用于存储全局变量，以便在程序的不同类中进行访问和调用。下面是对代码的分析：

1. **GlobalClass** 类声明在 **com.maoli.notepad\_maoli** 包下。
2. 类中定义了一系列成员变量，用于存储全局数据。这些成员变量包括：
  - **search**: 用于存储查找字符串。
  - **isCaseIgnored**: 用于表示查找时是否忽略大小写。
  - **replace**: 用于存储替换字符串。
  - **turnto**: 用于存储转到的位置。
  - **font**: 用于存储字体对象。
  - **fontColor**: 用于存储字体颜色。
  - **fontFamily**: 用于存储字体族。
  - **fontWeight**: 用于存储字体粗细。
  - **fontPosture**: 用于存储字体姿态。
  - **size**: 用于存储字体大小。

这些变量都是公共的（public），因此可以在程序的其他类中直接访问和修改它们的值。

通过在 **GlobalClass** 中存储这些全局变量，可以在程序的不同部分共享数据，避免了在各个类之间传递参数或创建多个实例的需要。这样可以更方便地管理和使用全局数据，并提高代码的可读性和维护性。

```
9 public class GlobalClass {
10     // 查找
11     8个用法
12     public String search;
13     // 查找时是否忽略大小写
14     12个用法
15     public boolean isCaseIgnored;
16     // 替换
17     6个用法
18     public String replace;
19     // 转到
20     2个用法
21     public int turnto;
22     // 字体
23     2个用法
24     public Font font;
25     // 字体颜色
```

## notepadController: (主要代码文件)

记事本应用的控制器类。它包含了一系列的 FXML 注解来定义图形用户界面元素以及事件处理方法。

这个控制器类继承自 **Stage** 类并实现了 **Initializable** 接口，说明它是一个 JavaFX 应用程序的主舞台，并且在初始化时会调用 **initialize** 方法。

下面是代码中定义的一些成员变量和注解：

- **@FXML** 注解用于注入 FXML 文档中定义的 GUI 元素，比如 **Stage**、**TextArea**、**Menu**、**MenuItem**、**ContextMenu** 等等。
- **private GlobalClass global;** 是一个类型为 **GlobalClass** 的私有成员变量，表示全局变量。
- **private Font font;** 是一个 **Font** 类型的私有成员变量，表示字体。
- **private double size = 20;** 是一个 **double** 类型的私有成员变量，表示字体大小，默认为 20。
- **private int zoomPercentage = 100;** 是一个 **int** 类型的私有成员变量，表示缩放百分比，默认为 100。
- **private int status = 1;** 是一个 **int** 类型的私有成员变量，表示记事本的状态，1 表示新建，2 表示已修改，3 表示已保存。
- **private String currentPath;** 是一个 **String** 类型的私有成员变量，表示当前文档的路径。
- .....

```

43 public class notepadController extends Stage implements Initializable
44 {
    9 个用法
45     @FXML private Stage stage;
    6 个用法
46     @FXML private Stage fontStage;
    6 个用法
47     @FXML private Stage turntoStage;
    28 个用法
48     private GlobalClass global;
    10 个用法
49     private Font font;
    12 个用法
50     private double size = 20;
    9 个用法
51     private int zoomPercentage = 100;
    71 个用法
52     @FXML public TextArea textArea;
    1 个用法
53     @FXML public Menu firstMenu;
    1 个用法

```



除了成员变量和注解外，代码还包含了一些方法用于处理不同的功能，比如新建文档、打开文档、保存文档等。这些方法通过@FXML 注解来指定与 FXML 文档中的 GUI 元素关联的事件处理方法，主要有以下方法：

1. **newFile()**: 当点击"新建"菜单项时调用该方法。首先会弹出一个确认对话框，询问用户是否要新建文档。如果用户点击"确定"按钮，则清空 **textArea** 中的文本内容，设置窗口标题为"无标题 - 记事本"，将 **status** 变量设置为 1，将 **currentPath** 变量置为 null。
2. **open()**: 当点击"打开"菜单项时调用该方法。创建一个文档选择器 **FileChooser**，设置对话框标题为"打开文档"，初始目录为"C:"。添加两个文档过滤器，一个用于文本文档(.txt)，一个用于所有文档(\*)。调用 **showOpenDialog** 方法显示文档选择对话框，并获取用户选择的文档。如果用户选择了文档，则将 **status** 变量设置为 2，将 **currentPath** 变量设置为选择的文档的绝对路径，设置窗口标题为选择的文档名。然后使用 **BufferedReader** 读取文档内容，并将内容显示在 **textArea** 中。
3. **save()**: 当点击"保存"菜单项时调用该方法。将 **status** 变量设置为 3。如果 **currentPath** 为 null，则调用 **saveAs()** 方法保存文档。否则，创建一个 **FileWriter** 对象，将 **textArea** 中的文本内容写入文档，然后刷新流和关闭文档。最后设置窗口标题为保存的文档路径加上" - 记事本"。
4. **saveAs()**: 当点击"另存为"菜单项时调用该方法。创建一个文档选择器 **FileChooser**，设置对话框标题为"保存文档"，初始目录为"C:"。添加一个文档过滤器，限制只能保存为文本文档(\*.txt)。调用 **showSaveDialog** 方法显示文档选择对话框，并获取用户选择的文档。如果用户选择了文档，则将 **status** 变量设置为 3，创建一个 **FileWriter** 对象，将 **textArea** 中的文本内容写入文档，然后刷新流和关闭文档。将 **currentPath** 变量设置为保存的文档的绝对路径，并设置窗口标题为当前文档路径加上" - 记事本"。
5. **pageSet()**: 当点击"页面设置"菜单项时调用该方法。创建一个 **PageFormat** 对象，并调用 **PrinterJob** 的 **pageDialog** 方法显示页面设置对话框。
6. **exit()**: 当点击"退出"菜单项时调用该方法。首先判断 **status** 变量的值，如果不等于 3，则弹出一个确认对话框询问用户是否要保存当前内容。根据用户的选择，执行相应的操作。如果 **currentPath** 为 null，则调用 **saveAs()** 方法保存文档并关闭窗口。否则，调用 **save()** 方法保存文档并关闭窗口。
7. **updateMenuInfo()**: 更新编辑菜单项的可选状态。根据 **textArea** 的状态，判断是否可撤销、可重做、可剪切、可复制、可粘贴、可删除。如果可撤销栈不为空，则将"撤销"菜单项设置为可选状态，否则设置为不可选状态。如果可重做栈不为空，则将"重做"菜单项设置为可选状态，否则设置为不可选状态。其他选项的可选状态也是根据 **textArea** 的状态进行设置。



8. **statusLabelUpdate()**: 更新状态栏的显示内容。根据 **status** 变量的值, 显示不同的状态信息。
9. **textAreaKeyPressed()**: 监听键盘按键抬起事件。当按下 Ctrl 键时, 将 **ctrlFlag** 变量设置为 true。当按下 Z 键且 **ctrlFlag** 为 true 时, 调用 **undo()** 方法执行撤销操作。当按下 Y 键且 **ctrlFlag** 为 true 时, 调用 **redo()** 方法执行重做操作。
10. **textAreaKeyReleased()**: 监听键盘按键释放事件。当释放 Ctrl 键时, 将 **ctrlFlag** 变量设置为 false。
11. **textAreaMousePressed()**: 监听鼠标点击事件。获取鼠标点击的位置信息。
12. **textAreaMouseReleased()**: 监听鼠标释放事件。获取鼠标释放的位置信息。
13. **textAreaMouseClicked()**: 监听鼠标点击事件。根据鼠标点击的位置信息, 判断是否显示右键菜单。
14. **textAreaScroll()**: 监听鼠标滚轮事件。根据鼠标滚轮的滚动方向, 调整 **size** 变量和 **zoomPercentage** 变量的值, 然后更新 **textArea** 中文本的字体大小。
15. **setFontName()**: 当选择字体名称时调用该方法。根据选择的字体名称, 更新 **font** 变量, 并更新 **textArea** 中文本的字体。
16. **setFontSize()**: 当选择字体大小时调用该方法。根据选择的字体大小, 更新 **size** 变量, 并更新 **textArea** 中文本的字体大小。
17. **undo()**: 执行撤销操作。从 **undoStack** 中取出最近的一个操作, 将其应用到 **textArea** 中, 并将该操作放入 **redoStack** 中。
18. **redo()**: 执行重做操作。从 **redoStack** 中取出最近的一个操作, 将其应用到 **textArea** 中, 并将该操作放入 **undoStack** 中。
19. **cut()**: 执行剪切操作。将选中的文本从 **textArea** 中剪切, 并将其放入系统剪贴板中。
20. **copy()**: 执行复制操作。将选中的文本复制到系统剪贴板中。
21. **paste()**: 执行粘贴操作。从系统剪贴板中获取文本, 并将其粘贴到 **textArea** 中的光标位置。
22. **delete()**: 执行删除操作。删除选中的文本。



```

114 public String getCurrentPath() { return currentPath; }
118
119 public void mainStage(Stage stage) { this.stage = stage; }
123
124 @Override
125 public void initialize(URL location, ResourceBundle resources)
126 {
127     global = (GlobalClass) mainApplication.globals.get("global");
128     mainApplication.controllers.put(this.getClass().getSimpleName(), this);
129
130     font = Font.font("微软雅黑", FontWeight.NORMAL, FontPosture.REGULAR, size);
131     textArea.setFont(font);
132     statusLabel.setText(" | 第 " + 1 + " 行, 第 " + 1 + " 列 " + " | " + zoomPercentage + "% |");
133
134     updateMenuInfo();
135     fourthMenu_Status.setSelected(true);
136     // 监控鼠标点击
137     textArea.setOnMouseClicked(event -> {
138         statusLabelUpdate();
139         updateMenuInfo();
140     });
141
142     // 监控鼠标滚轮
143     textArea.setOnScroll(event -> {
144         if(event.isControlDown() && event.getDeltaY() < 0)
145             zoomout();
146         else if(event.isControlDown() && event.getDeltaY() > 0)
147             zoomin();
148     });
149 }

```

总体来说，该代码实现了记事本应用的基本功能，包括新建、打开、保存文档，编辑文本内容，设置字体，查找替换等功能。

### fontController:

字体选择器的控制器，用于控制字体、字形、大小和颜色的选择。以下是对代码的详细分析：

1. 导入必要的包和类。
2. 声明并初始化全局变量 **global** 和 **controller**，分别表示全局类和主控制器类。
3. 声明和初始化 FXML 中定义的各个控件。
4. 定义常量 **ChineseShowText**、**EnglishShowText** 和 **NumberShowText**，分别为用于显示的中文、英文和数字文本。
5. 声明 **font** 变量，用于存储选择的字体。
6. 声明 **sizeArray** 变量，并初始化一个可观察列表，用于存储所有的字体大小。

```

22 public class fontController implements Initializable {
23     11 个用法
24     public GlobalClass global;
25     4 个用法
26     public notepadController controller;
27     9 个用法
28     @FXML private ListView<String> fontSelector;
29     8 个用法
30     @FXML private ListView<String> glyphSelector;
31     6 个用法
32     @FXML private ListView<String> sizeSelector;
33     5 个用法
34     @FXML private ComboBox<String> fontColorSelector;
35     2 个用法
36     @FXML private TextField fontTextField;
37     2 个用法
38     @FXML private TextField glyphTextField;
39     2 个用法
40     @FXML private TextField sizeTextField;
41     4 个用法
42     @FXML private TextArea fontShowTextArea;
43     1 个用法
44     private static final String ChineseShowText = "测试文本";
45     1 个用法
46     private static final String EnglishShowText = "XMU Notepad";
47     1 个用法
48     private static final String NumberShowText = "1234567890";

```

7. 实现 **initialize** 方法，该方法在控制器初始化时被调用。

- 获取全局类和主控制器类的实例。
- 获取本地图形环境。
- 初始化字体、字形、大小和颜色的可观察列表，并设置到相应的控件中。
- 绑定字体选择器、字形选择器和大小选择器的选择项属性与对应的文本框。
- 设置字体选择器、字形选择器、大小选择器和颜色选择器的默认选择项。
- 调用 **myFontSet()** 方法进行字体设置。
- 添加选择项监听器，当选择项改变时调用 **myFontSet()** 方法。

8. 实现 **myFontSet()** 方法，用于设置字体。

- 初始化默认字体大小为 20。
- 根据选择的字体大小，找到对应的字体大小值。
- 根据选择的字形，设置字体的粗体和斜体属性。
- 获取选择的字体颜色。
- 更新全局类的相关属性。
- 设置显示字体的文本区域的字体和颜色。
- 设置显示字体的文本区域的文本内容。

9. 实现 **fontInterfaceCancelButtonPressed()** 方法，当取消按钮被按下时调用，用于关闭字体选择界面。

10. 实现 **fontInterfaceConfirmButtonPressed()** 方法，当确认按钮被按下时调用，用于确认字体选择并关闭字体选择界面。



索操作。

- 获取搜索和替换的字符串。
- 更新全局类的相关属性。
- 调用主控制器类的 `previousSearch()` 方法。

8. 实现 `replace()` 方法，当替换按钮被按下时调用，用于执行替换操作。

- 获取搜索和替换的字符串。
- 更新全局类的相关属性。
- 调用主控制器类的 `replace()` 方法。

9. 实现 `replaceAll()` 方法，当替换全部按钮被按下时调用，用于执行替换全部操作。

- 获取搜索和替换的字符串。
- 更新全局类的相关属性。
- 调用主控制器类的 `replaceAll()` 方法。

该代码通过 JavaFX 提供的控件和事件处理机制，实现了搜索和替换功能。用户可以输入要搜索的字符串和要替换的字符串，并通过点击按钮来执行搜索、替换、替换全部等操作。同时，代码通过全局类和主控制器类的实例，实现了与主界面的交互，将搜索和替换的相关信息传递给主控制器类进行处理。

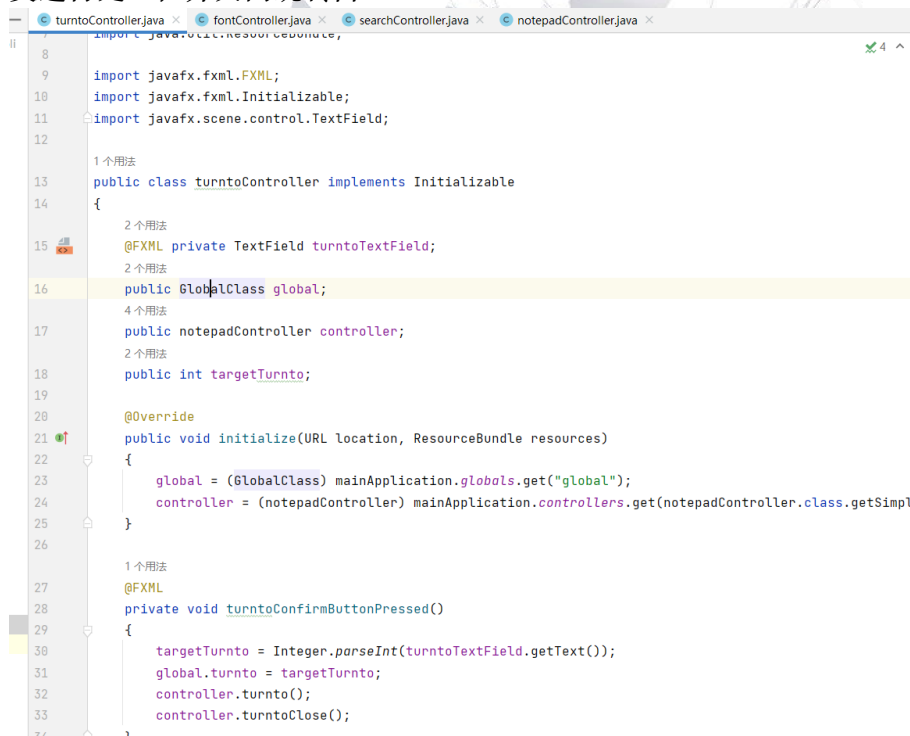
```
1 个用法
46  @FXML
47  private void previousSearch()
48  {
49      searchString = searchTextField.getText();
50      replaceString = replaceTextField.getText();
51      global.search = searchString;
52      global.replace = replaceString;
53      global.isCaseIgnored = !IgnoreCaseBox.isSelected();
54      controller.previousSearch();
55  }
56
57  // 替换功能
58  1 个用法
59  @FXML
60  private void replace()
61  {
62      searchString = searchTextField.getText();
63      replaceString = replaceTextField.getText();
64      global.search = searchString;
65      global.replace = replaceString;
66      global.isCaseIgnored = !IgnoreCaseBox.isSelected();
67      controller.replace();
68
69  // 替换全部功能
```

## turntoController:

跳转到指定行的控制器，用于在文本编辑器中跳转到用户指定的行数。以下是对代码的详细分析：

1. 导入必要的包和类。
2. 声明并初始化 FXML 中定义的 **TextField** 控件，用于用户输入目标行数。
3. 声明并初始化全局变量 **global** 和 **controller**，分别表示全局类和主控制器类。
4. 声明整型变量 **targetTurnto**，用于存储目标行数。
5. 实现 **initialize** 方法，该方法在控制器初始化时被调用。
  - 获取全局类和主控制器类的实例。
6. 实现 **turntoConfirmButtonPressed()** 方法，当确认按钮被按下时调用，用于执行跳转操作。
  - 获取用户输入的目标行数，并转换为整型。
  - 更新全局类的 **turnto** 属性。
  - 调用主控制器类的 **turnto()** 方法，实现跳转到指定行的操作。
  - 调用主控制器类的 **turntoClose()** 方法，关闭跳转窗口。
7. 实现 **turntoCancelButtonPressed()** 方法，当取消按钮被按下时调用，用于关闭跳转窗口。
  - 调用主控制器类的 **turntoClose()** 方法，关闭跳转窗口。

该代码通过 JavaFX 提供的控件和事件处理机制，实现了在文本编辑器中跳转到指定行的功能。用户可以在文本框中输入目标行数，并通过点击确认按钮来执行跳转操作。同时，代码通过全局类和主控制器类的实例，实现了与主界面的交互，将目标行数传递给主控制器类进行处理，并关闭跳转窗口。

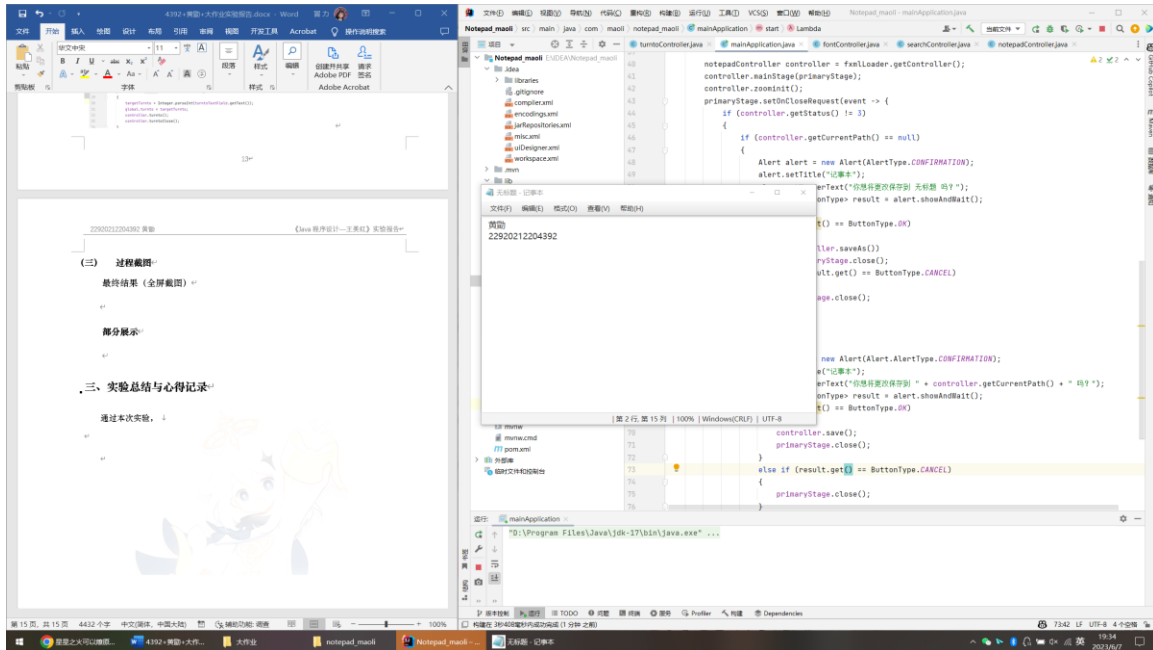


```
1  import java.util.ResourceBundle;
2
3  8
4
5  9
6  import javafx.fxml.FXML;
7  import javafx.fxml.Initializable;
8  import javafx.scene.control.TextField;
9
10 11
12 12
13 1 个用法
14  public class turntoController implements Initializable
15  {
16
17  2 个用法
18  @FXML private TextField turntoTextField;
19
20  2 个用法
21  16
22  public GlobalClass global;
23
24  4 个用法
25  public notepadController controller;
26
27  2 个用法
28  public int targetTurnto;
29
30
31  19
32  @Override
33  public void initialize(URL location, ResourceBundle resources)
34  {
35
36  23
37      global = (GlobalClass) mainApplication.globals.get("global");
38      controller = (notepadController) mainApplication.controllers.get(notepadController.class.getSimpleName());
39  }
40
41
42  26
43
44  1 个用法
45  27
46  @FXML
47  private void turntoConfirmButtonPressed()
48  {
49
50  29
51      targetTurnto = Integer.parseInt(turntoTextField.getText());
52      global.turnto = targetTurnto;
53      controller.turnto();
54      controller.turntoClose();
55  }
56
57  34
```

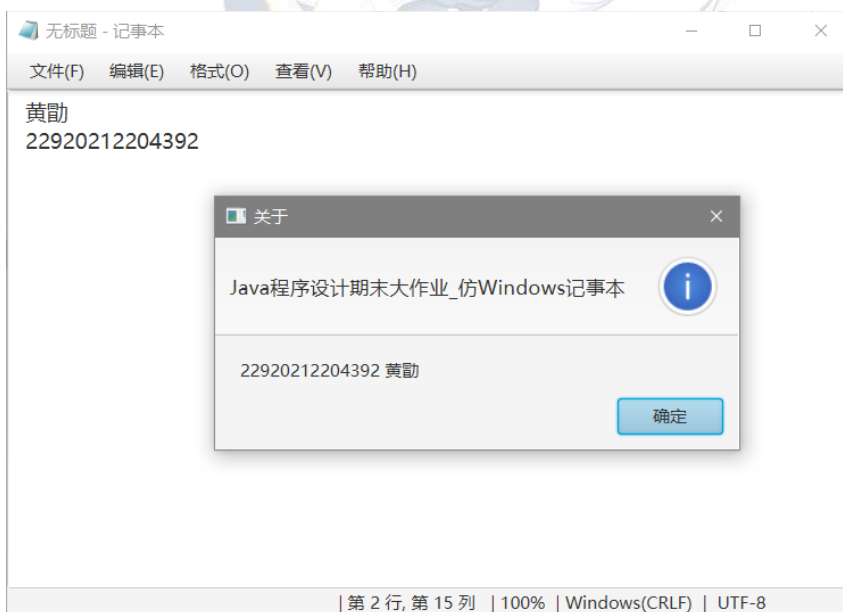


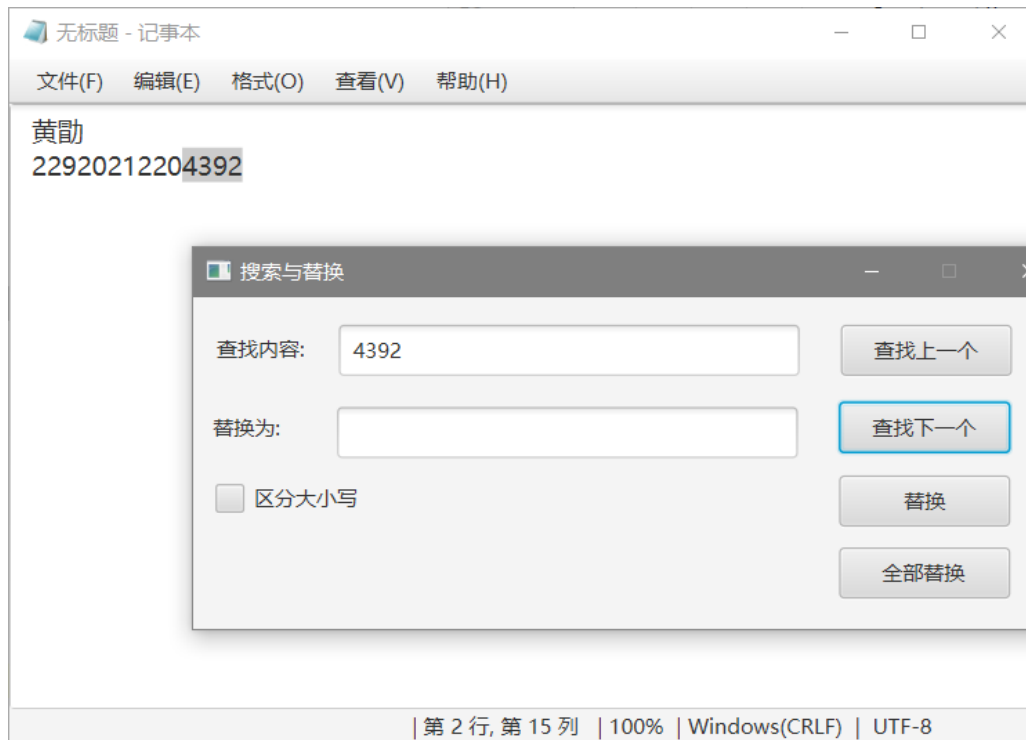
### (三) 过程截图

最终结果（全屏截图）



部分展示

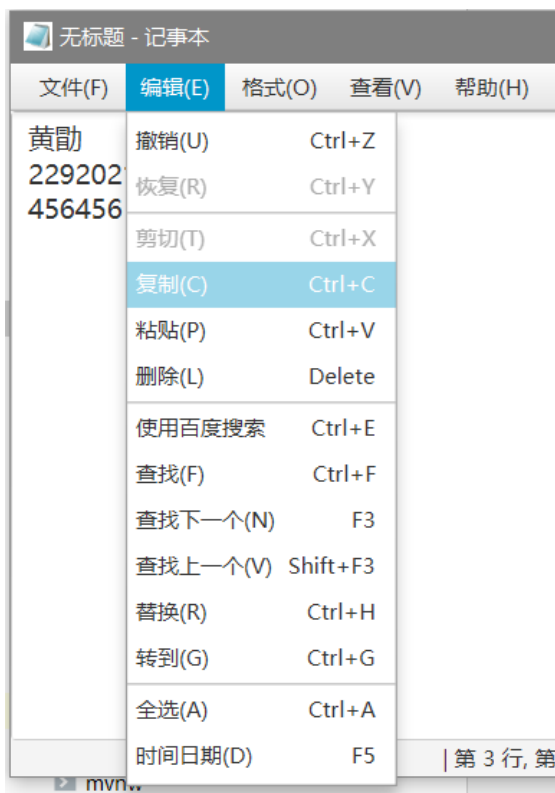




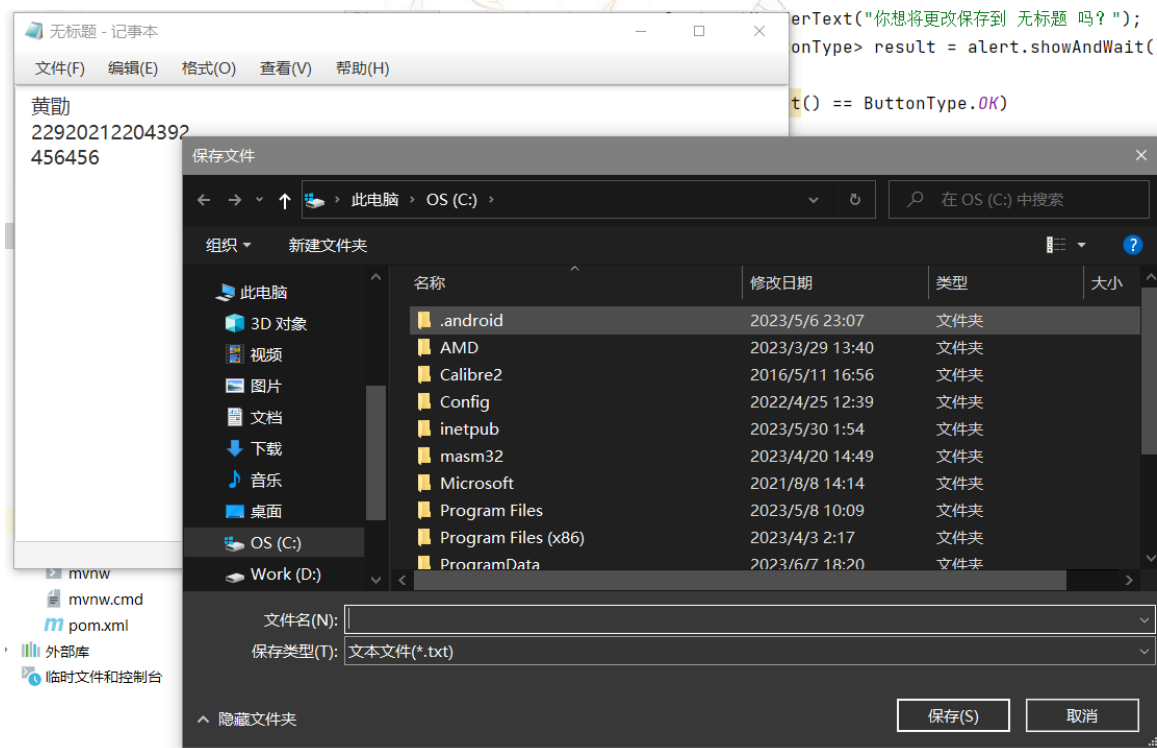
实现把 123 替换为 456



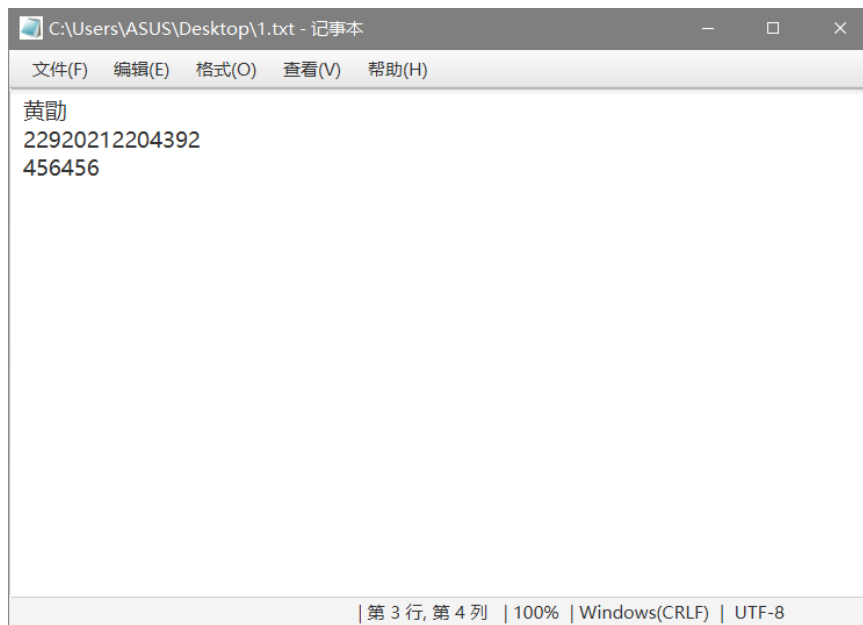
编辑菜单，部分显示灰色不可操作，达到要求



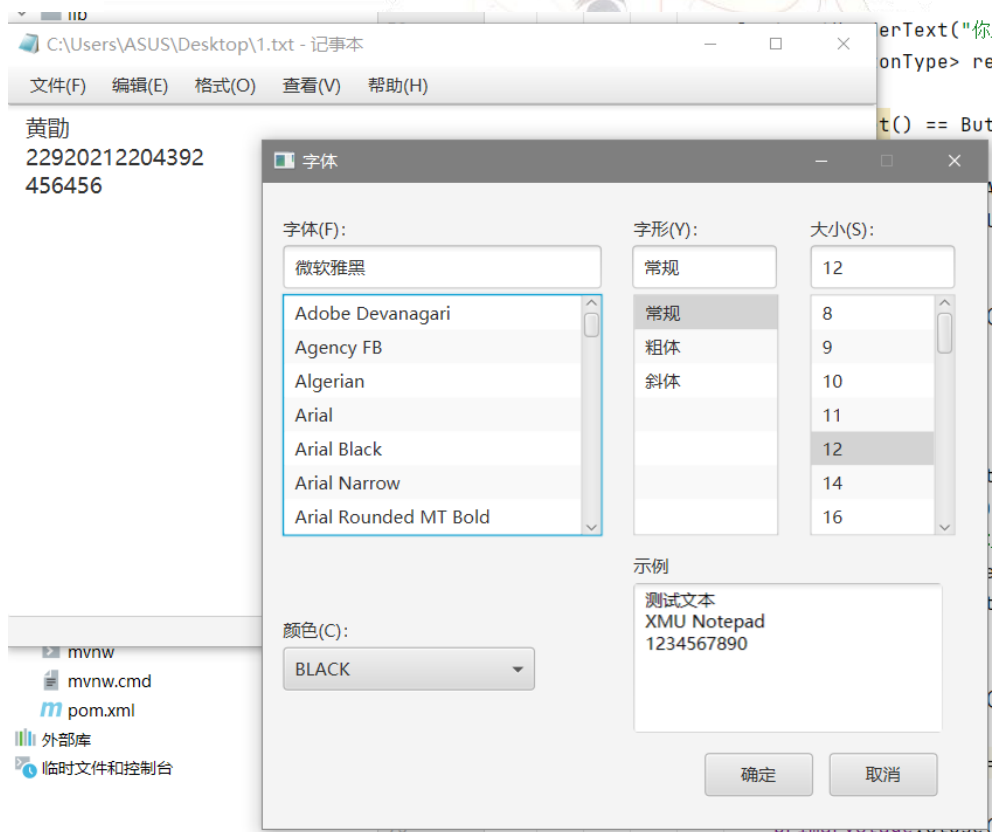
## 保存文件



可以实现打开：



字体：



转到行:

### 三、实验总结与心得记录

在完成 Java 记事本应用程序的大作业后，我的第一感觉是成就感，经过一周时间的反复调试和编写，我对整个实验过程有了一些心得和体会。以下是我在完成这个大作业时所学到的一些经验和观点：

1. **项目规划和设计的重要性:** 在开始实验之前，进行良好的项目规划和设计是非常关键的。这包括确定所需功能和特性、设计应用程序的界面和交互流程，并考虑如何组织代码和模块化的设计。一个清晰的项目规划和设计有助于避免后期的混乱和困惑，提高开发效率。
2. **分模块开发和测试:** 将整个应用程序划分为模块并分别进行开发和测试是一种有效的方法。这样可以提高代码的可维护性和可测试性，每个模块都可以独立测试和调试，确保其功能正常，并且在集成时更容易排除错误。
3. **良好的代码组织和命名规范:** 为了确保代码的可读性和可维护性，采用良好的代码组织结构和命名规范是非常重要的。使用有意义的变量和方法命名，按照一定的代码风格和约定编写代码，可以使代码更易于理解和修改，并且减少后期的错误和调试时间。

4. **错误处理和异常处理:** 在编写应用程序时,要考虑各种可能的错误和异常情况,并相应地进行处理。合理的错误处理和异常处理机制可以提高程序的稳定性和可靠性,确保程序在遇到异常情况时不会崩溃或产生不可预料的结果。

5. **用户界面设计和用户体验:** 记事本应用程序的用户界面设计和用户体验对于用户的满意度和使用感受至关重要。要注意设计简洁直观的界面,合理布局和组织功能,提供友好的交互方式,使用户能够方便地操作和使用应用程序。

6. **测试和调试:** 测试和调试是开发过程中不可或缺的环节。编写适当的单元测试和集成测试,并进行反复的测试和调试,以确保应用程序的各个功能和模块正常工作,并能够正确处理各种情况和输入。

7. **持续学习和改进:** 在完成这个大作业的过程中,我不断学习和探索新的知识和技术。我发现通过查阅文档、阅读相关书籍,可以获得更多的灵感和解决问题的思路。持续学习和改进是成为一个优秀的开发者的关键。

总的来说,完成 Java 记事本应用程序的大作业是一次非常有益的实验。通过这个实验,我不仅巩固了 Java 编程的基础知识,还学习了 JavaFX 界面开发、文档操作、异常处理、事件处理等方面的知识和技能。这个实验也提高了我在项目规划、代码组织和团队协作方面的能力。我相信这些经验和技能对我的软件开发之路将会有很大的帮助!

黄勛

2023.6