哈尔滨工业大学（深圳）

面向对象的软件构造导论
实验指导书

实验五 Swing 和多线程

2023 春

目录

1. 实验目的 3
2. 实验环境 3
3. 实验内容（4 学时） 3
4. 实验步骤 4
    4.1 Java Swing 4
        4.1.1 Swing GUI Designer 4
        4.1.2 JTable 组件 9
        4.1.3 CardLayout 卡片布局 12
    4.2 Java 多线程编程 15
        4.2.1 使用 Runnable 接口实现多线程 16
        4.2.2 继承 Thread 类实现多线程 16
5. 实验要求 17

1. 实验目的

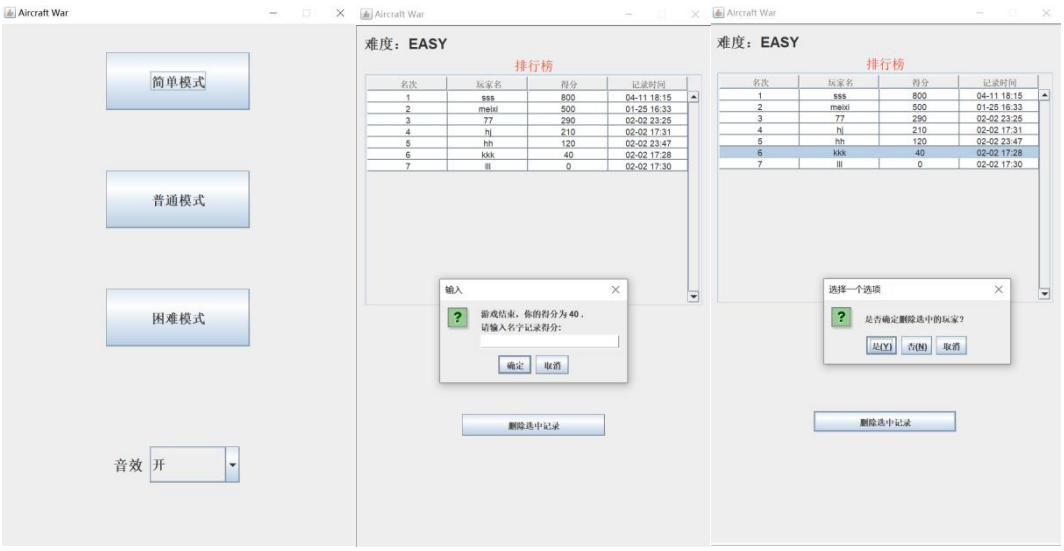
1. 熟悉 Java 图形界面程序设计的基本方法，掌握 Java Swing 中的容器、常用组件和布局管理器的使用。
2. 理解 Java 事件处理机制，掌握事件处理机制的基本用法。
3. 理解 Java 多线程的概念和生命周期，掌握多线程的实现方法。

2. 实验环境

1. Windows 10
2. IntelliJ IDEA 2022.3.2
3. Java 11

3. 实验内容（4 学时）

（1）使用 Java Swing 类库完成飞机大战游戏中的难度选择、音效开关界面。
（2）使用 Java Swing 类库完成得分排行榜界面，显示该难度的玩家得分排名。并可与玩家进行交互，输入玩家姓名后可记录该局得分，并可删除任一玩家的历史得分。
界面与下图接近即可：



（3）使用 Runnable 接口实现多线程，完善火力道具功能。
（4）继承 Thread 类实现多线程，完成游戏背景音乐、子弹击中敌机、炸弹爆炸、道具生效、Boss 机出场、游戏结束时的音效控制。

4. 实验步骤

4.1 Java Swing

Swing 是 Java 为图形界面应用开发提供的一组工具包，是 Java 基础类的一部分。使用 Swing 来开发图形界面比 AWT 更加优秀，因为 Swing 是一种轻量级组件，它采用纯 Java 实现，不再依赖于本地平台的图形界面，所以可以在所有平台上保持相同的运行效果，对跨平台支持比较出色。除此之外，Swing 提供了比 AWT 更多的图形界面组件，因此可以开发出美观的图形界面程序。

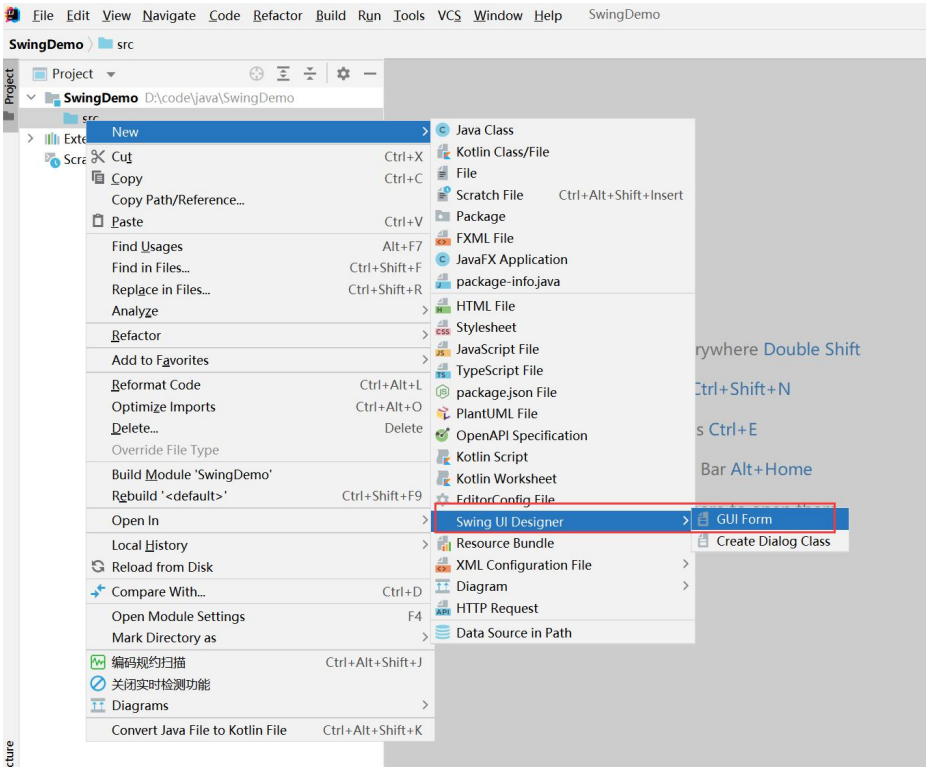
Swing 组件都采用 MVC（Model-View-Controller，即模型-视图-控制器）的设计，实现 GUI 组件的显示逻辑和数据逻辑的分离，以提供更多的灵活性。

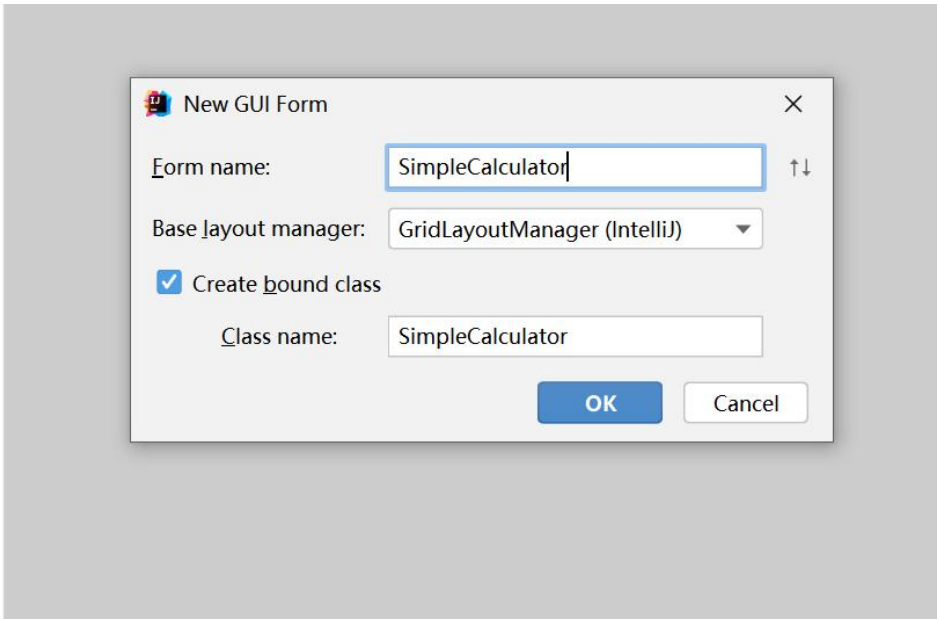
Swing 包含了构建图形界面（GUI）的各种组件，如：窗口、标签、按钮、文本框等。Swing 组件在 javax.swing.*包下，类名均以 J 开头，例如：JFrame、JLabel、JButton 等。

4.1.1 Swing GUI Designer

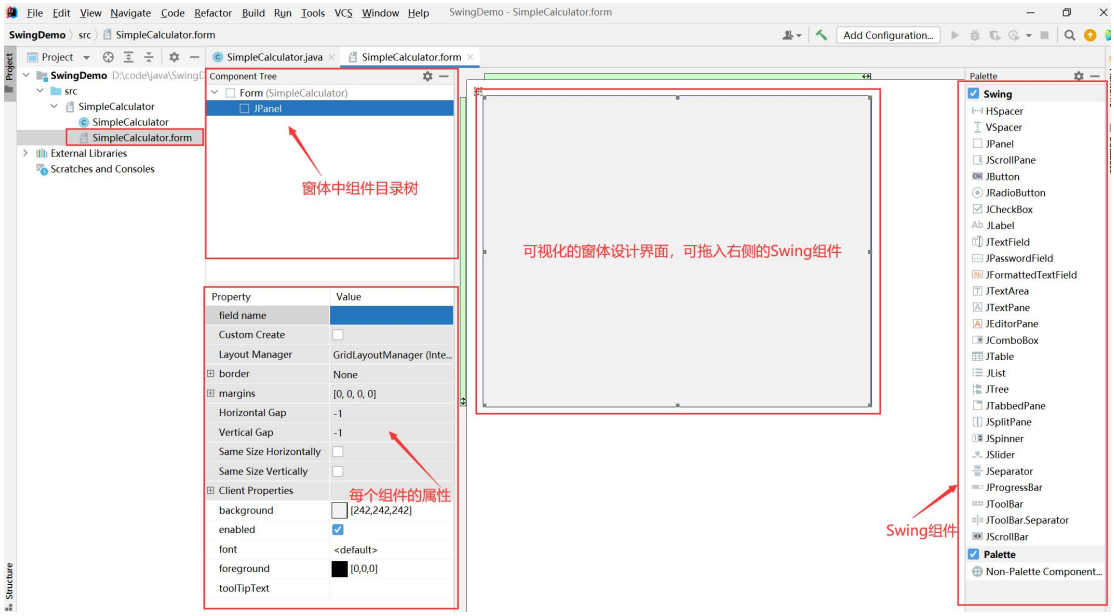
飞机大战游戏中的难度选择、音效开关、得分排行榜等功能需要使用图形化界面与用户进行交互。IntelliJ IDEA 提供的 Swing GUI Designer 可以方便的进行图形界面编程，有兴趣的同学还可以尝试功能更强大的 JFormDesigner 插件（非开源）。

(1) 在 IntelliJ IDEA 中创建一个 Java Project （本例中 project 名为 SwingDemo），右键 src 目录，新建一个 GUI Form，命名为：SimpleCalculator。这里我们设计一个简单的计算器，输入两个数字，点击运算按钮可显示相应的运算结果。

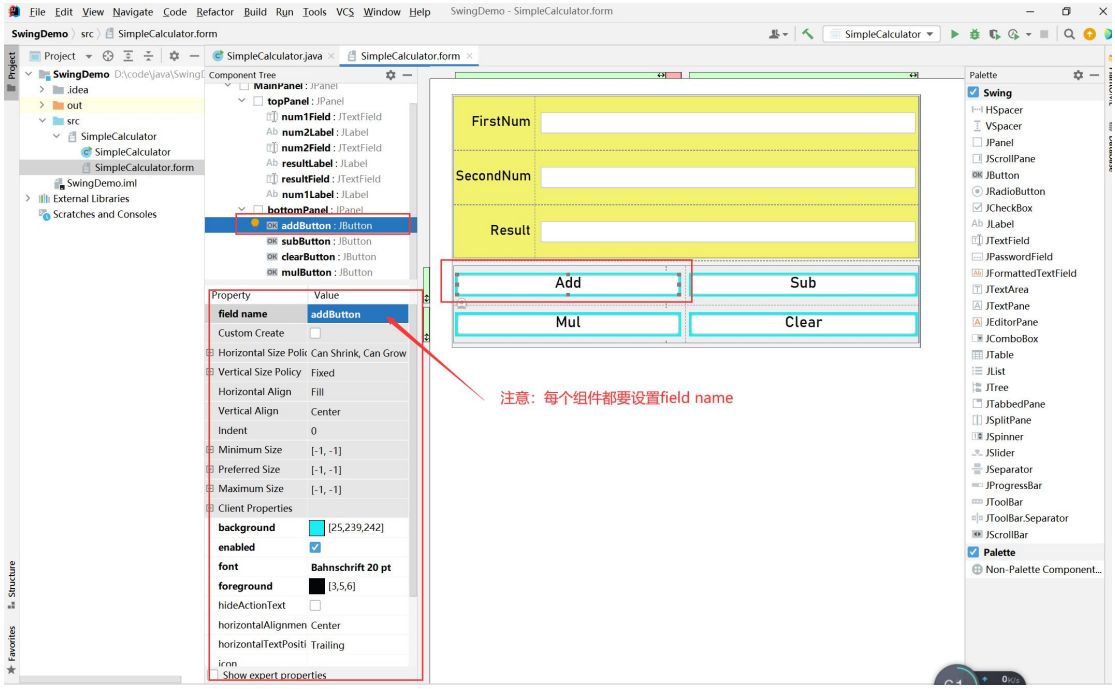




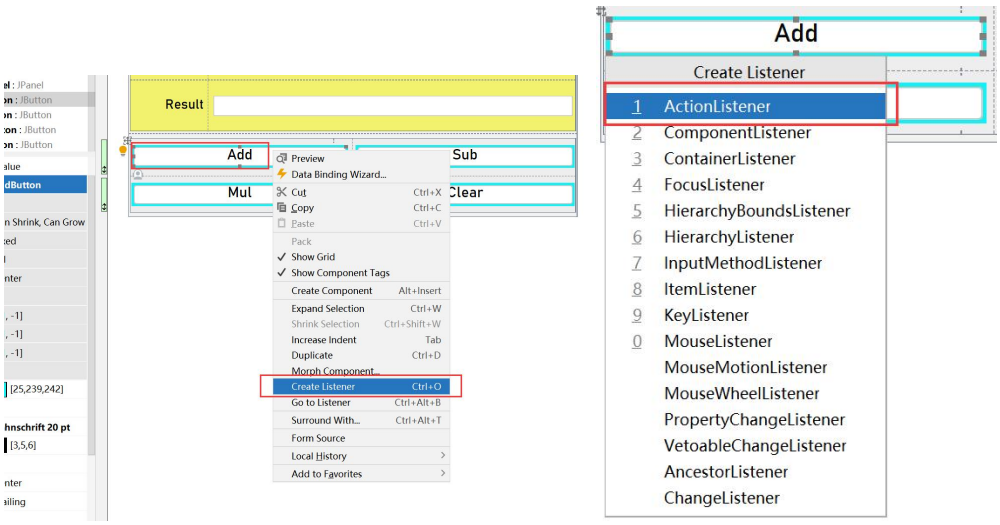
(2) 自动生成两个文件：SimpleCalculator.form 用于图形界面设计，而 SimpleCalculator.java 是其对应的 Java 文件，可以操作组件对象和运行，双击 SimpleCalculator.form 可以看到可视化的窗体设计界面。



(3) 在窗口设计界面中拖入所需要的 Swing 组件并合理布局。点击每个组件可方便的设置其属性值，如颜色，字体，大小等。注意每个组件都要设置 field name。鼠标点击右键选择 Preview 选项可查看窗体设计的效果。



(4) 添加按钮事件，右键 Add 按钮，选择 Create Listener 选项，为其添加 ActionListener。



在 SimpleCalculator.java 中会自动生成如下代码，在 actionPerformed()函数体中添加处理该按钮点击事件的代码。

```
        addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
            //添加事件处理代码
            }
        });
```

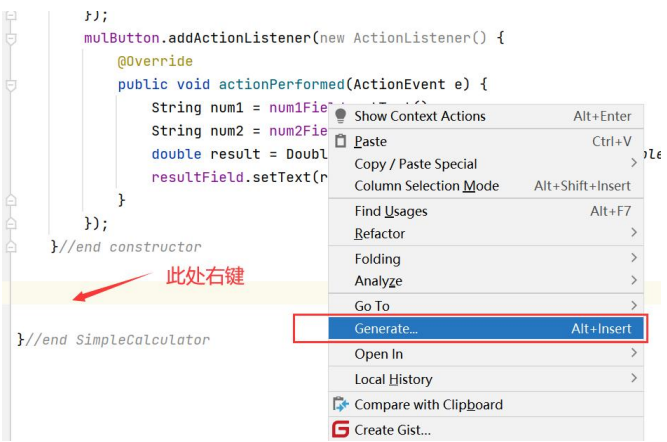
添加其他按钮事件：

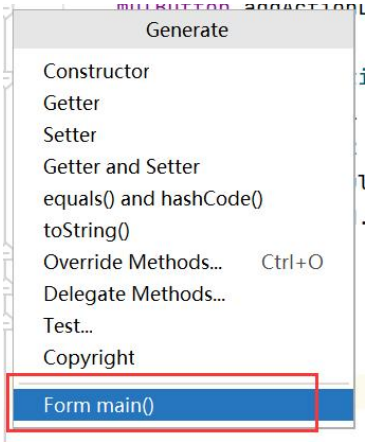
SimpleCalculator.java

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SimpleCalculator {
    private JPanel MainPanel;
    private JPanel topPanel;
    private JPanel bottomPanel;
    private JLabel num1Label;
    private JLabel num2Label;
    private JLabel resultLabel;
    private JTextField num2Field;
    private JTextField resultField;
    private JTextField num1Field;
    private JButton addButton;
    private JButton clearButton;
    private JButton subButton;
    private JButton mulButton;


    public SimpleCalculator() {
        addButton.addActionListener(new ActionListener() {
            @Override
```

```
            public void actionPerformed(ActionEvent e) {
                String num1 = num1Field.getText();
                String num2 = num2Field.getText();
                double result = Double.parseDouble(num1) + Double.parseDouble(num2);
                resultField.setText(result + "");
            }
        });
        subButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String num1 = num1Field.getText();
                String num2 = num2Field.getText();
                double result = Double.parseDouble(num1) - Double.parseDouble(num2);
                resultField.setText(result + "");

            }
        });
        clearButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                num1Field.setText("");
                num2Field.setText("");
                resultField.setText("");
            }
        });
        mulButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String num1 = num1Field.getText();
                String num2 = num2Field.getText();
                double result = Double.parseDouble(num1) * Double.parseDouble(num2);
                resultField.setText(result + "");
            }
        });
    }
}
```

(5) 添加 Form Main 函数。

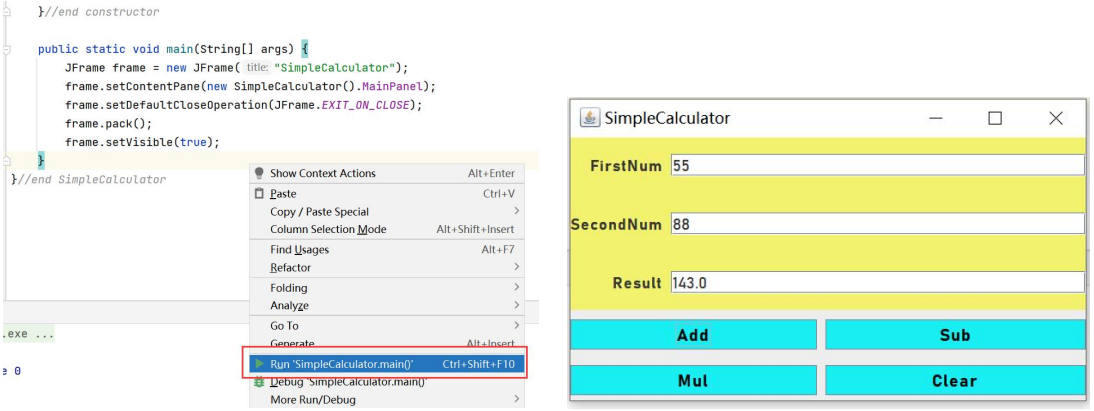




在 SimpleCalculator.java 中会自动生成如下代码：

```
public static void main(String[] args) {
        JFrame frame = new JFrame("SimpleCalculator");
        frame.setContentPane(new SimpleCalculator().MainPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
}
```

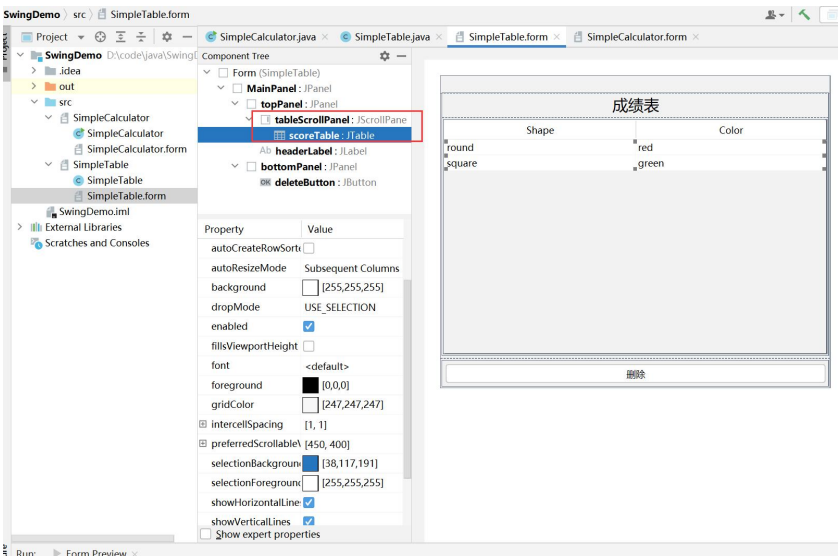
运行程序：



4.1.2 JTable 组件

JTable 是将数据以表格的形式显示给用户看的一种组件，它包括行和列，其中每列代表一种属性，例如：学号、姓名和成绩。而每行代表的是一个实体，例如一个学生。

(1) 新建一个 GUI Form，命名为 SimpleTable。这里我们设计一个简单的表格显示某个班级的学生成绩。添加所需组件（JScrollPanel 和 JTable）并合理布局。



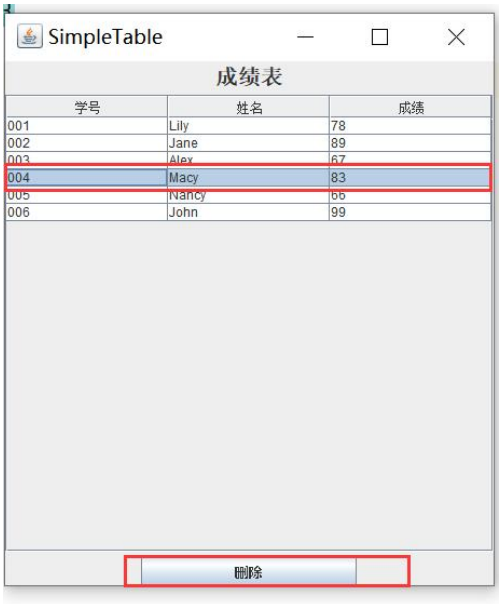
(2) JTable 表格与数据通过 TableModel 分离，JTable 并不存储自己的数据，而是从表格模型那里获取它的数据。在构造函数中创建 DefaultTableModel 对象，装载数据。

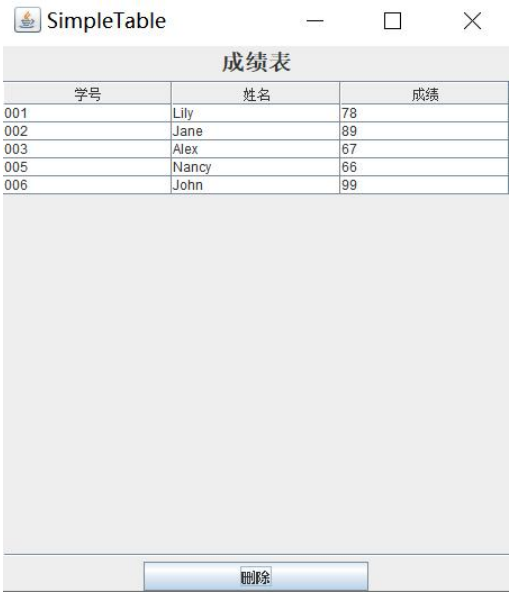
```
public SimpleTable() {

        String[] columnName = {"学号","姓名","成绩"};
        String[][]tableData={{"001","Lily","78"},{"002","Jane","89"},{"003","Alex","67"},
                {"004","Macy","83"},{"005","Nancy","66"},{"006","John","99"}};

        //表格模型
        DefaultTableModel model = new DefaultTableModel(tableData, columnName){
            @Override
            public boolean isCellEditable(int row, int col){
                return false;
            }
        };

        //从表格模型那里获取数据
        scoreTable.setModel(model);
        tableScrollPanel.setViewportView(scoreTable);

    }
```

(3) 添加“删除”按钮事件和 Form Main 函数。

```
        deleteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int row = scoreTable.getSelectedRow();
                System.out.println(row);
                if(row != -1){
                    model.removeRow(row);
                }
            }
        });
```

运行程序：





完整代码参考：

SimpleTable.java

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SimpleTable {
    private JPanel MainPanel;
    private JPanel bottomPanel;
    private JPanel topPanel;
    private JScrollPane tableScrollPanel;
    private JLabel headerLabel;
    private JTable scoreTable;
    private JButton deleteButton;

    public SimpleTable() {

        String[] columnName = {"学号","姓名","成绩"};
        String[][]tableData={{"001","Lily","78"},{"002","Jane","89"},{"003","Alex","67"},
                {"004","Macy","83"},{"005","Nancy","66"},{"006","John","99"}};

        //表格模型
        DefaultTableModel model = new DefaultTableModel(tableData, columnName){
            @Override
            public boolean isCellEditable(int row, int col){
                return false;
            }
        };

        //JTable 并不存储自己的数据，而是从表格模型那里获取它的数据
        scoreTable.setModel(model);
        tableScrollPanel.setViewportView(scoreTable);
```

```
        deleteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int row = scoreTable.getSelectedRow();
                System.out.println(row);
                if(row != -1){
                    model.removeRow(row);
                }
            }
        });
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("SimpleTable");
        frame.setContentPane(new SimpleTable().MainPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

4.1.3 CardLayout 卡片布局

CardLayout 布局管理器以时间而非空间来管理它里面的组件，能够让多个组件共享同一个显示空间。它将加入容器的所有组件看成一叠卡片（每个卡片其实就是一个组件），组件叠在一起，初始时显示该空间中第一个添加的组件，通过 CardLayout 类提供的方法可以切换该空间中显示的组件。CardLayout 类的常用构造函数和方法如下所示：

方法名称	功能
CardLayout()	组件距容器左右边界和上下边界的距离为缺省值 0 个像素。
CardLayout(int hgap,int vgap)	组件距容器左右边界和上下边界的距离为指定值。
void first(Container parent)	翻转到容器的第一张卡片
void next(Container parent)	翻转到容器的下一张卡片
void previous(Container parent)	翻转到容器的前一张卡片
void last(Container parent)	翻转到容器的最后一张卡片
void show(Container parent, String name)	显示指定卡片

（1）新建一个 CardLayoutDemo 类，定义并初始化卡片布局对象 cardLayout 以及使用它的容器 cardPanel。

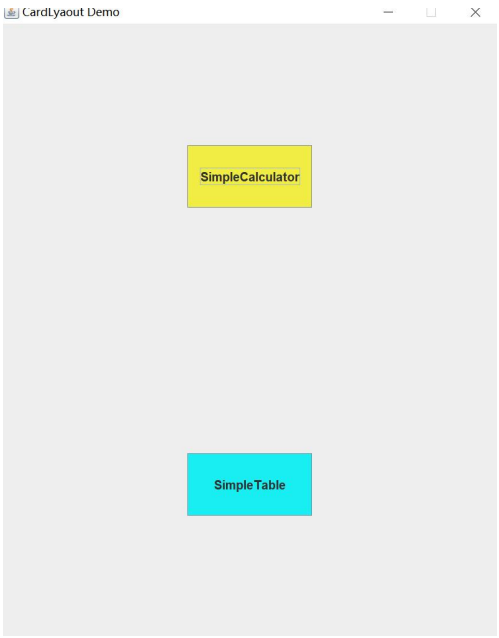
```
CardLayout cardLayout = new CardLayout(0,0);
JPanel cardPanel = new JPanel(cardLayout);
```

（2）新建窗体 JFrame，并将容器 cardPanel 加入其中。

```
JFrame frame = new JFrame("CardLyaout Demo");
frame.setSize(800, 1024);
frame.setResizable(false);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.add(cardPanel);
```

（3）新建一个 GUI Form，命名为 StartMenu。这里我们设计一个只有两个按钮的简单页面，用以实现多个页面的切换。添加所需组件（JButton）并合理布局。新建 StartMenu 页面，加入容器 cardPanel 并显示。

```
StartMenu start = new StartMenu ();
cardPanel.add(start.getMainPanel());
frame.setVisible(true);
```



（4）为 StartMenu 的两个按钮添加事件，在 actionPerformed()函数体中将点击按钮要跳转的页面加入容器 cardPanel，并使用 cardLayout 实现页面切换。

```
simpleCalculatorButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        CardLayoutDemo.cardPanel.add(new SimpleCalculator().getMainPanel());
        CardLayoutDemo.cardLayout.last(CardLayoutDemo.cardPanel);
    }
});
```

完整代码参考：

CardLayoutDemo.java

```
public class CardLayoutDemo {

    static final CardLayout cardLayout = new CardLayout(0,0);
    static final JPanel cardPanel = new JPanel(cardLayout);

    public static void main(String[] args) {

        JFrame frame = new JFrame("CardLyaout Demo");
        frame.setSize(800, 1024);
        frame.setResizable(false);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.add(cardPanel);

        StartMenu start = new StartMenu();
        cardPanel.add(start.getMainPanel());
        frame.setVisible(true);
    }
}
```

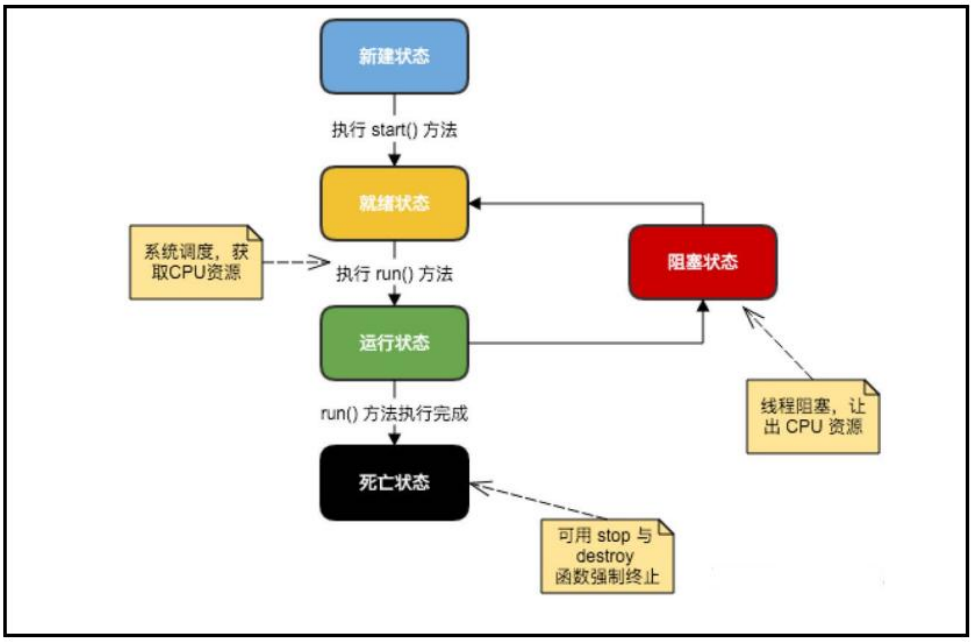
StartMenu.java

```
public class StartMenu {


    private JPanel mainPanel;
    private JButton simpleCalculatorButton;
    private JButton simpleTableButton;

    public StartMenu() {
        simpleCalculatorButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                CardLayoutDemo.cardPanel.add(new SimpleCalculator().getMainPanel());
                CardLayoutDemo.cardLayout.last(CardLayoutDemo.cardPanel);
```

```
            }
        });
        simpleTableButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                CardLayoutDemo.cardPanel.add(new SimpleTable().getMainPanel());
                CardLayoutDemo.cardLayout.last(CardLayoutDemo.cardPanel);
            }
        });
    }

    public JPanel getMainPanel() {
        return mainPanel;
    }

}
```

4.2 Java 多线程编程

飞机大战游戏中，火力道具生效、音效多处控制等功能需要用多线程来完成。Java 给多线程编程提供了内置的支持。 一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。线程是一个动态执行的过程，它也有一个从产生到死亡的过程。
下图显示了一个线程完整的生命周期。



Java 提供了多种创建线程的方法， 以下介绍两种常用方式：使用 Runnable 接口和通过继承 Thread 类本身来实现多线程。

4.2.1 使用 Runnable 接口实现多线程

可以使用外部类、静态内部类、匿名类等多种方式实现 Runnable 接口。由于 Runnable 是一个函数式接口（只包含一个抽象方法的接口），我们可以用一个 lambda 表达式创建一个实例，避免匿名内部类定义过多，再从这个 Runnable 实例构造一个 Thread 对象启动线程。

```
public class RunnableTest {
    public static void main(String[] args) {
        Runnable r = () -> {
            try {
                for (int i = 0; i < 3; i++) {
                    System.out.println("【" + Thread.currentThread().getName() + "】
                    线程执行，当前的循环次数：" + i);
                    Thread.sleep(2000);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        };
        // 启动线程
        new Thread(r, "线程 1").start();
        new Thread(r, "线程 2").start();
    }
}
```

4.2.2 继承 Thread 类实现多线程

创建一个线程的第二种方法是创建一个新的类，该类继承 Thread 类，然后创建一个该类的实例。继承类必须重写 run() 方法，该方法是新线程的入口点。它也必须调用 start() 方法才能执行。该方法尽管被列为一种多线程实现方式，但是本质上也是实现了 Runnable 接口的一个实例。本实验已提供 MusicThread 类，该类继承 Thread 类，重写了 run()方法，用于启动音频播放。

```
    @Override
    public void run(){
        InputStream stream =new ByteArrayInputStream(samples);
        play(stream);
    }
```

使用方法：

```
public class ThreadTest {
```

```
    public static void main(String[] args) {
        new MusicThread("src/bgm.wav").start();
    }
}
```

在飞机大战游戏中，还需实现循环播放音频、停止播放音频的功能，请根据需求修改 MusicThread 类。

5. 实验要求

实验课前，预习并熟悉 Swing 中的容器和常用组件的用法。理解 Java 事件处理和多线程机制。

本次实验提交版本需完成以下功能：

- ✓ 游戏开始显示难度选择和音效设置界面，根据玩家选择显示相应难度的游戏地图。
- ✓ 若音效开启，游戏中循环播放游戏背景音乐，游戏结束后停止播放。子弹击中敌机、炸弹爆炸、道具生效、游戏结束时有相应的音效。Boss 敌机出场时循环播放其背景音乐，坠毁或游戏结束后停止播放。
- ✓ 火力道具生效时，英雄机由直射切换为散射弹道并持续一段时间，结束后恢复直射状态。
- ✓ 游戏结束后显示得分排行榜界面，要求可记录玩家该局得分，并可删除玩家历史得分。

注意：本实验无需细化三种不同游戏难度，替换地图即可，实验六进一步完善。

✧ **提交内容**

包括：

① 整个项目压缩包（整个项目压缩成 zip 包提交）