Weather App

操作方式

- 3个step均实现，版本信息如下(使用gradle管理项目)

```
------------------------------------------------------------
Gradle 8.5
------------------------------------------------------------

Build time:   2023-11-29 14:08:57 UTC
Revision:     28aca86a7180baa17117e0e5ba01d8ea9feca598

Kotlin:       1.9.20
Groovy:       3.0.17
Ant:          Apache Ant(TM) version 1.10.13 compiled on January 4 2023
JVM:          21 (Oracle Corporation 21+35-LTS-2513)
OS:           Windows 11 10.0 amd64
```

- 打开压缩包并解压后可以看到主目录下有一个`weather.jar`的可执行文件，在运行之前建议关闭网络，这样可以用最少的步骤完成测试，可以直接点击运行，也可以通过命令行的方式运行(`java -jar .\weather.jar`)，打开之后界面会在1到2秒内完成更新，如果网络未连接，则会弹出提示框，提示框可关闭，这时界面上的内容都是第一步中所要求的内容，在搜索框中输入一个不为空的字符，点击搜索按钮，会看到弹出提示框，关闭，恢复网络连接，可以看到界面就会被更新，这时去测试城市搜索功能，输入一个可行的城市，比如`Berlin`，然后点击搜索按钮，会看到下方会出现3个选项，随便选一个，可以看到界面被更新，并且那三个选项也消失，最后，可以随意输入一个搜不到的城市，会看到效果和内容为空的时候是一样的
    1. 关闭网络连接
    2. 打开程序
    3. 关闭弹出的提示框
    4. 在搜索框中输入一个不为空的字符，点击搜索按钮
    5. 关闭弹出的提示框
    6. 连接网络(界面自己会更新)
    7. 输入城市名，点击搜索按钮(界面更新)
    8. 输入搜不到的城市名，点击搜索按钮(显示`unavailable`并会自己消失)

架构与设计

- 整体思路就是分为5个模块，分别是标题，城市搜索，当前天气，空气污染，未来天气用5个继承JPanel的类去实现，与功能相对应，并且为了提高效率，为每一个类实现Runnable接口，这样多线程运行，效率会好一些，此外还有1个类，负责去维护他们共同使用的一些信息，并且这些获取和修改这些信息都被写为synchronized，这样就会避免出现一些问题，最后来一个类负责去管理这5个类，包括启动他们和安排他们的位置
- 当前天气，空气污染，未来天气这三个的实现都十分相似，先在构造函数中初始化各个内容，初始化的信息就是第一步中给出的信息，然后写一个update方法，负责调用API去从网络获取信息并更新内容，然后run方法就是调用update，当然需要先判断一些条件是否满足在调用，同时捕获异常并进行处理，run函数如下，这是当前天气中的，空气污染，未来天气并不会再网络异常的时候弹出提示框，那样就出现三个提示框，不是很合理，可以看到判断的条件有`DataPool.isCurUpdate()`和`DataPool.isNetConnect()`，前者意思就是当前天气信息是否需要更新，后者是网络，一开始二者均为true

```
    public void run() {
        while (true) {
            try {
                if (DataPool.isCurUpdate() && DataPool.isNetConnect()) {
                    update();
                    DataPool.setCurUpdate(false);
                } else {
                    // Thread.sleep(1000);
                }
            } catch (ConnectException ce) {
                displayNetworkErrorDialog("网络连接异常，请检查网络设置");
            } catch (SocketTimeoutException ste) {
                displayNetworkErrorDialog("网络连接超时，请稍后重试");
            }  catch (Exception e) {
                System.out.println(e);
            }
        }
    }
```

- 标题的实现比较简单，就是从数据池中获取信息并使用以及计算时间，但是除此之外，它也维护了一些其他类的信息，可以看出，它会每隔一分钟左右更新当前天气，空气污染，未来天气的更新状态，这样那三个类就会实现自动更新，以及如果没有网络连接，就每隔5秒去判断网络是否重连，并更新网络信息，这样再网络恢复后，那三个类就会识别到并更新界面

```
    public void run() {
        while (true) {
            try {
                updateTitle();
                Thread.sleep(1000);
                cnt++;
                cnt %= 60;
                if (cnt == 0) {
                    DataPool.setAirUpdate(true);
                    DataPool.setCurUpdate(true);
                    DataPool.setFutUpdate(true);
                }
                if (!DataPool.isNetConnect() && cnt % 5 == 0) {
                    if (isNetworkConnected())
                        DataPool.setNetConnect(true);
                }
            } catch (Exception e) {

            }
        }
    }
```

- 城市搜索的功能，主要就是两个事件监听器的处理逻辑，就是比较繁杂，没有什么难的地方，run函数也是十分简单，就是一个死循环，但是我在开启线程的时候，为它开了高优先级，这样就可以保证两个事件监听器的监听效果

```
        new Thread(titlePanel).start();
        new Thread(curweatherPanel).start();
        new Thread(futureweatherPanel).start();
        new Thread(airPanel).start();
        new Thread(cityPanel).start();
        cityPanel.setPriority(Thread.MAX_PRIORITY);
```

- 最后的主类就很简单了，主要是使用GridBagLayout去管理他们的位置信息

```
class WeatherAppPanel extends JPanel {
    private TitlePanel titlePanel;
    private CurWeather curweatherPanel;
    private FutWeather futureweatherPanel;
    private AirPollution airPanel;
    private CitySearch cityPanel;
    // 添加其他面板或组件的引用

    public WeatherAppPanel() {
        setLayout(new GridBagLayout());
        setBackground(Color.CYAN);
        GridBagConstraints myGrid = new GridBagConstraints();
        myGrid.insets = new Insets(5, 5, 5, 5);

        titlePanel = new TitlePanel();
        myGrid.gridx = 0;
        myGrid.gridy = 0;
        myGrid.gridwidth = 6;
        myGrid.anchor = GridBagConstraints.NORTH;
        myGrid.fill = GridBagConstraints.HORIZONTAL;
        add(titlePanel, myGrid);
        // 添加其他面板或组件的初始化代码

        new Thread(titlePanel).start();
        new Thread(curweatherPanel).start();
        new Thread(futureweatherPanel).start();
        new Thread(airPanel).start();
        new Thread(cityPanel).start();
        cityPanel.setPriority(Thread.MAX_PRIORITY);
    }
}
```

- main函数也就只需要把主面板加进来就ok了

```
    public static void main(String[] args) {
        JFrame frame = new JFrame("Weather App");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setMinimumSize(new Dimension(1200, 900));

        // 创建和添加主面板
        WeatherAppPanel mainPanel = new WeatherAppPanel();
        frame.getContentPane().add(mainPanel);
        frame.pack();

        frame.setVisible(true);
    }
```