CHRISTOPHER NEWPORT UNIVERSITY

DEPARTMENT OF PHYSICS, COMPUTER SCIENCE & ENGINEERING

CPSC 280: INTRODUCTION TO SOFTWARE ENGINEERING

* Assignment 4: GUI Programming *

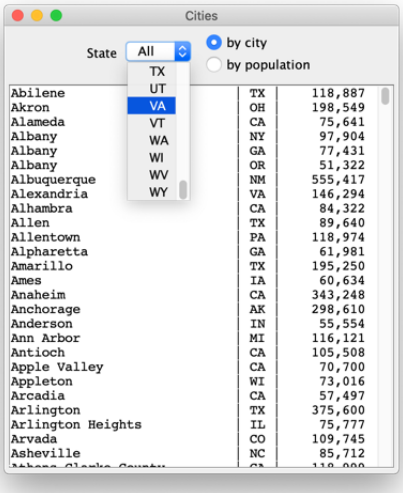
Instructor: Dr. Roberto A. Flores

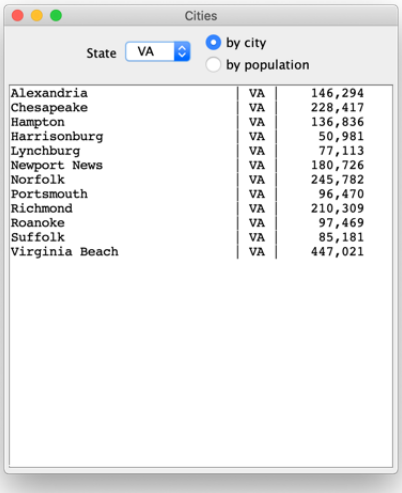
Goal

Practice programming GUI in Java Swing.

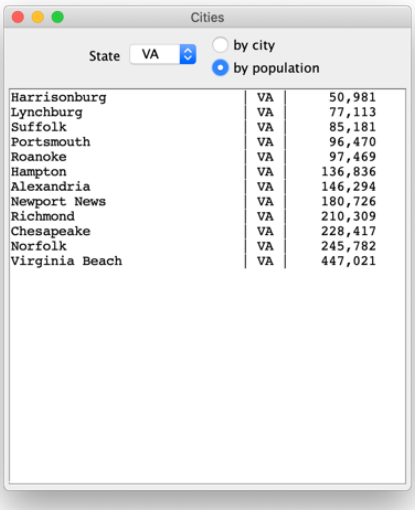
Cities

Write a program displaying a list of cities (each with a name, state and population) in a GUI frame. Users choose a state from the combo box (e.g., VA left) to see the cities in that state only (right).



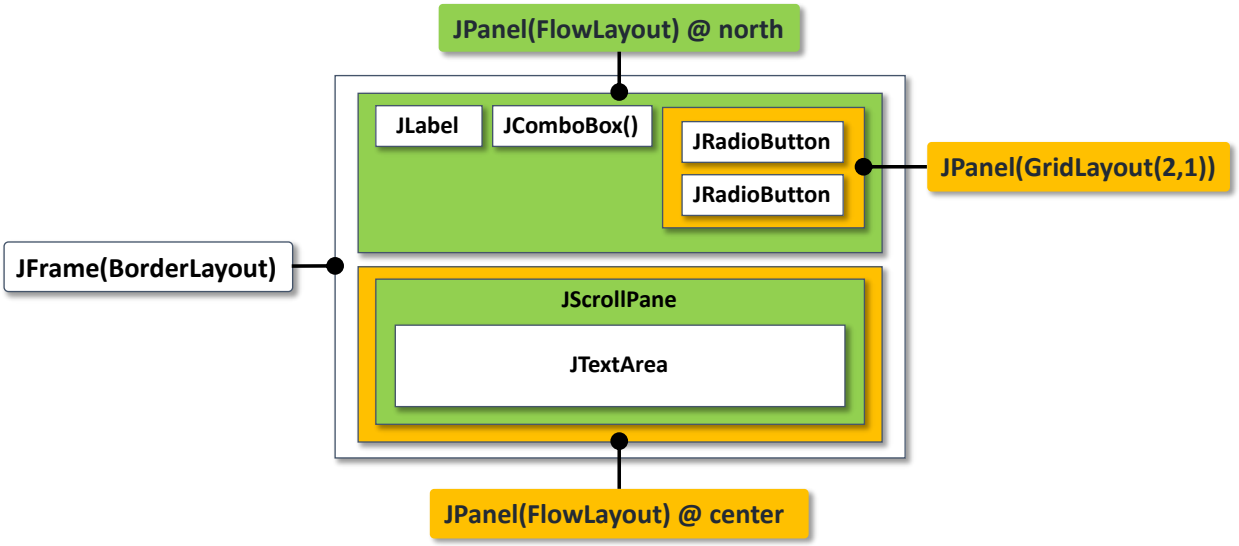


Also, users click on radio buttons to sort displayed cities either by name or by population (below).



Implementation

Write a class named *Cities* that subclasses from *JFrame*. This class has a *main* method (creating a new object of our class and making it visible) and a default constructor (customizing the contents of our frame). The design of our frame is shown below.



As shown above, our frame has a border layout with two panels:

- **North**: A panel (with a flow layout) containing a label "State", a combo box with state abbreviations, and a panel (with a grid layout of 2 rows and 1 column) containing two radio buttons "by city" and "by population" (in that order). The combo box has the values below (in the order shown):

  ```
  { "All","AK","AL","AR","AZ","CA","CO","CT","DC","DE","FL","GA","HI","IA","ID","IL",
  "IN","KS","KY","LA","MA","MD","ME","MI","MN","MO","MS","MT","NC","ND","NE","NH",
  "NJ","NM","NV","NY","OH","OK","OR","PA","RI","SC","SD","TN","TX","UT","VA","VT",
  "WA","WI","WV","WY" }
  ```

  Radio buttons are logically grouped (use *ButtonGroup*) and thus are mutually exclusive.

- **Center**: A panel (with a flow layout) containing a scroll pane wrapping up a text area where city data is displayed. The scroll pane has a preferred size of 400-by-400 pixels, and the text area has a Courier font and size 14. These can be initialized as follows:

  ```
  JTextArea   area   = new JTextArea();
  JScrollPane scroll = new JScrollPane( area );
  area  .setFont( new Font( "Courier", Font.PLAIN, 14 ));
  scroll.setPreferredSize( new Dimension( 400, 400 ));
  ```

  The text area displays city data collected from a CSV file "cities.csv" (given to you). Each line in the file has data of one city, in the format "name, state, population". Note that there is a blank space after each comma; thus use ", " (comma blank space) as a delimiter to split these values. **Do not hard code** the path to the CSV file; rather place it in the same folder as your *Cities.java* file and use the following command to locate it (note: you can initialize a *Scanner* with this input stream):

```
InputStream is = getClass().getResourceAsStream( "cities.csv" );
```

Text areas have a method *setText()*receiving a string with the entire contents to display. You will need to assemble all city data in one string before using this method. Note: in the figures shown in this description, the city data is neatly displayed in "columns" using the formatting string

```
"%-28s | %s | %,10d%n"
```

where the first value is city name (a string, left aligned in 28 spaces), followed by its state (a string, taking 2 spaces by default), its population (an integer, right aligned in 10 spaces and using commas for thousand and millions) and an end-of-line (i.e., "%n") at the end of each city string; that way the text area will display a city per line. The last city in the list should not have a trailing end-of-line.

The frame is centered on the screen, it is packed (i.e., use *pack()* –do **not** use *setSize()*– to automatically get the preferred size of its components) and it is disposed on close.

Implement listeners for the combo box and radio buttons, each of which update the contents of the text area whenever the user changes their selected values.

When the program starts our frame has the following state:

- The combo box has option "All" selected;
- The radio button "by city" is selected; and (accordingly)
- The text area displays the data of all cities sorted by city name.

Note that when sorting by city name, cities with the same name (e.g. Springfield, IL and Springfield, MA) are sorted by state; and when sorting by population, cities of equal population are sorted by city name.

Use the test file *CitiesTest.java* to validate your implementation.

Groups

This assignment is done individually.

You must follow the "Empty Hands" policy described in the syllabus. Review the class syllabus or contact me if you're unsure about this policy.

Submission & Grading

Your work must be submitted by the due date to the **Gitlab repository** (https://gitlab.pcs.cnu.edu/) **given to you by the instructor**.

Important notes about the git repository:

- It must be formatted as a **Gradle** project.
- Gradle's **src/main/java** folder must contain your *solution files*.
- Use the *build.gradle* file given to you on scholar.
- Use a *.gitignore* file generated by <http://gitignore.io>. Generate a file customized for OSX, Windows, Gradle, Eclipse and Java (all of them); and make sure it is effectively used in your repository. Execute the following git commands in your repo to enable the new *.gitignore* file:
    - git rm -r --cached .
    - git add .

Grades for this assignment are based on the test cases given to you.

Up to 20 points could be deducted for non-compliance with the assignment description, e.g., static analysis errors/warnings (warning suppression is not allowed), failed Gitlab submission, Gradle misconfiguration, incorrect *.gitignore*.

Javadoc is not required.

Solutions to test cases must not be hardcoded.

•••