

COMP 1020 Winter 2016 Assignment 3

Due Date: Sunday, March 20, before midnight

1 MATERIAL COVERED

- Class hierarchies: inheritance and polymorphism

2 NOTES AND INSTRUCTIONS—PLEASE FOLLOW *EXACTLY*

- Follow the programming standards posted on the course website to avoid losing marks.
- You must complete the “Blanket Honesty Declaration” checklist on the course website, before you can submit any assignment.
- To submit the assignment you will upload the required files, as specified for each question, to the Dropbox for Assignment 3 on the course website.
- In this assignment you will create some simple GUI ‘widgets’ using the StdDraw module. There are no output files to hand in. The markers will compile and execute your code. To be eligible to earn full marks, your Java programs must compile and run as downloaded, without requiring any modifications.
- The questions in this assignment are of unequal size. Question 1 is the largest one. Question 2 is very small (4 new classes, but they require only 3 lines of executable code between them!).
- Hand in **only and exactly** the files specified in this assignment, in the specified format. Failure to do so will result in lost marks.

3 QUESTION 1: A HOME-MADE GUI (GRAPHICAL USER INTERFACE)

In this question, you will complete a set of 6 classes that will implement, in a StdDraw graphics window, four standard widgets that appear in graphical user interfaces (GUIs): buttons, checkboxes, radio buttons, and text boxes. (See the image on the next page.) Each will be written as a separate class (Button, Checkbox, RadioButton, and TextBox). These will all be subclasses of the superclass GUIelement. There will also be a “collection” class GUIgroup which will hold a list of GUIelement objects (and it will be a subclass of GUIelement, too). The two classes GUIelement and Checkbox will be given to you. You will write the other four.

This will give you some idea of how real-life GUIs work, and how objects and OOP are used to create them.

Input box 1

Input box 2

Output box

Left Right

☒ L choice 1 ☒ R choice 1

☐ L choice 2 ☐ R choice 2

☐ L choice 3 ☐ R choice 3

☐ L option ☐ R option

The instructions that follow may seem long, but they give a very detailed description of what each class does or needs, and it should make it easy to write them.

3.1 Look at the **supplied** superclass `GUIelement.java`. All other classes in this question will be subclasses of this one. This is an *abstract* class. There are 6 instance variables common to every `GUIelement` object:

- A set of four **double** values that define the rectangular area in the `StdDraw` window that will contain this object: `xCentre`, `yCentre`, `halfWidth`, and `halfHeight`.
- A **String** variable `text` that will hold the contents of a text box, the label in a button, or the label beside a radio button or checkbox. Every subclass has a different use for this variable, but almost all of them will need a **String** for something.
- A **boolean** variable `highlighted` which will indicate whether or not this item is selected/active/highlighted (the exact meaning depends on the type of object).
- All 6 of these variables are **protected** variables because the subclasses will constantly require access to them (and it's much easier than making them private and writing 12 get/set methods).

There are the following public methods that every `GUIelement` will need:

- Two constructors: one which will initialize all 6 variables, and another with no parameters which leaves them at default values (0/null/false).
- Accessor (get) methods that will return the values of the `text` and `highlighted` variables. There is no need for mutator (set) methods.
- A **void** `draw()` method. Every `GUIelement` object will use this to draw itself in some appropriate way. The superclass method erases everything in its rectangle (draws

a filled white rectangle over it), and then draws a thin black outline around it. Some subclasses will find this useful. Others will override it.

- d. A **boolean** `handleClick(double x, double y)` method. This method is called whenever the user presses the mouse button at the point (x,y) in the `StdDraw` window. This method returns **true** if the point is within the rectangle for this object, and **false** otherwise. (The subclasses will all take advantage of this, but they must also take appropriate action to respond to the click.)
- e. A **boolean** `handleCharTyped(char c)` method. This method is called when the user types a character on the keyboard. This method always returns **true** if this `GUIElement` object knows what to do with a typed character, and **false** otherwise. The superclass method always returns **false**, which is the default action, since it has no appropriate way to use a character. Only textboxes will override this.

3.2 Look at the **supplied** class `Checkbox.java` which is a subclass of `GUIElement`.

- a. It has a constructor which accepts four parameters (**xc,yc,size,title**) where **xc** and **yc** are the coordinates of the centre of the box, **size** is both the half-height and half-width of the box (checkboxes are usually square), and **title** is the label that is shown to the right of the box.
- b. The `draw()` method uses the superclass `draw()` method to draw the checkbox outline. It also draws its **text** (the title) just to the right of the box (leaving a little space between the box and the title). If the box is **highlighted**, it draws an X in the box (using two short lines).
- c. The `handleClick(double x, double y)` method uses the superclass `handleClick` method to determine whether or not the mouse click was in this checkbox. If it was, the **highlighted** flag for this checkbox is toggled (changed from **true** to **false** or from **false** to **true**), the checkbox is re-drawn to show its new status, and **true** is returned (indicating “I handled it”).

3.3 Implement the `GUIGroup` class, which should be a subclass of `GUIElement` (this will allow smaller subgroups to be a part of a larger group, which is a technique used in all real GUI systems).

- a. Use an `ArrayList` to implement a list of `GUIElements`.
- b. Write a constructor (no parameters) which creates an empty list.
- c. Write a **void** `addElement(GUIElement e)` method which will add a new element to the end of the list.
- d. Write a **void** `draw()` method, overriding the superclass method, which applies the `draw()` method to every `GUIElement` in the list.
- e. Write two methods **boolean** `handleClick(double x, double y)` and **boolean** `handleCharTyped(char c)`, overriding the superclass methods. Each should apply

that same method to every `GUIElement` in the list. As soon as one of the list elements returns `true` this method should stop and return `true`. If none of the elements in the list ever returns `true`, then this method should return `false`.

- f. Write a `resetRadioButtons()` method which will apply a `reset()` method to all `RadioButton` objects in this list. (But *only* to the `RadioButton` objects – no other kind of `GUIElement` will have that method at all.)

3.4 Implement the `RadioButton` class as a subclass of `GUIElement`.

- a. This class should have one extra instance variable of type `GUIgroup` which keeps track of which group of `GUIElements` this button is a part of. This is needed because radio buttons are not independent, but function as a group. When one of them is selected, it automatically unselects the others in its group.
- b. Write a constructor which accepts six parameters (`xc`, `yc`, `radius`, `title`, `hilite`, `g`) where `xc` and `yc` are the coordinates of the centre of the button, `radius` is the radius of the button (radio buttons are always circular), `title` is the label that should be shown to the right of the button, and `g` is the `GUIgroup` that this button is a part of. Use the superclass constructor appropriately.
- c. Write a `draw()` method which will override the superclass `draw()` method – no rectangle is wanted this time. Instead, it should draw a hollow circle (solid white with a thin black outline) with the proper centre and radius. It should also draw its `text` (the title) just to the right of the button (leave a little space between the button and the title, and don't forget that there is a `StdDraw.textLeft` method). If the button is `highlighted`, then fill in the button with a smaller solid black circle inside the outer one.
- d. Write a `void reset()` method which will set `highlighted` to `false` and redraw the button. This method will be called whenever some other radio button in the same group as this one is selected.
- e. Write a `handleClick(double x, double y)` method which will use the superclass `handleClick` method to determine whether or not the mouse click was in this button. If it was, this method should apply the `resetRadioButtons()` method to the `GUIgroup` that this button is a part of. This will reset all radio buttons in this group (including this one). Then the `highlighted` flag for this button should be changed to `true`, the button should be re-drawn to show its new status, and `true` should be returned (indicating "I handled it"). Otherwise `false` should be returned.

3.5 Implement the `Button` class

- a. Write a constructor with five parameters (`xc, yc, hw, hh, title`) where the first four define the rectangular area for this button, and `title` is the label that will appear within it. Use the superclass constructor appropriately. The button should not be highlighted.

- b. Write a `draw()` method which will override the superclass `draw()` method. If the button is highlighted, it should be a solid `LIGHT_GRAY` rectangle, otherwise it should be a solid `WHITE` rectangle. Either way, it should have a black outline. The `text` should be drawn inside the button, in the centre.
- c. Write a `handleClick(double x, double y)` method which will use the superclass `handleClick` method to determine whether or not the mouse click was in this button. If it was, the button should be highlighted, and redrawn. It should remain that way until the user releases the mouse button, at which point it should be unhighlighted and redrawn again. Then `true` should be returned (indicating “I handled it”). Otherwise `false` should be returned. A small `Utilities` class will be provided in the file `Utilities.java`, and it will contain a `Utilities.waitMouseUp()` method which can be used to wait until the mouse button is released.

3.6 Implement the `TextBox` class which will provide both input boxes into which the user can type input, and output boxes which simply display information.

- a. This class should have one extra instance variable: a `boolean` value indicating whether or not this is an input box that can process characters typed by the user, or an output box that only displays text.
- b. A constructor with parameters `(xc, yc, hw, hh, txt, inp)` where the first four are the usual specifications for the rectangular area, `txt` is the text to be displayed in the box, `highlighted` is `false`, and `inp` is a `boolean` value indicating whether or not this is an input box.
- c. Write a `draw()` method which will use the superclass `draw` method to draw the rectangular outline of this box, and then display the `text` inside it. The text should be left-justified, starting a small distance from the left side of the box.
- d. Write a `handleClick(double x, double y)` method. As usual, the superclass `handleClick` method will tell you whether or not the click is inside this box, and whether you should return `true` or `false`. If the click is inside this box, and if it is an input box, then the click should clear the existing text from the box (make it an empty string) and set `highlighted` to `true` (meaning the box is now ready to accept input). If it isn't an input box, then clicks will be ignored.
- e. Write a `handleCharTyped(char c)` method. This is the only type of object that can handle a typed character. If the box is an input box, and it is currently `highlighted`, then it should handle the character as follows: If `c` is not the RETURN character (`'\n'`), then append the typed character to the end of the text in the box and redraw it. If `c` is the RETURN character, set `highlighted` to `false`. In either case, return `true` since it has now handled the character. If it is not an input box, it should return `false` and ignore the character.
- f. Finally, write a `displayText(String s)` method which will change the text in the box to `s`, and redraw it.

3.7 Run the supplied `TestA3Q1` test program to ensure that all of your classes are working properly.

3.8 HAND IN: Your 4 files `Button.java`, `GUIgroup.java`, `RadioButton.java`, and `TextBox.java`. Do *not* hand in `TestA3Q1`, `Utilities`, `GUIelement`, `Checkbox`, or `StdDraw`.

4 QUESTION 2: ADDING VERY SIMPLE EVENT HANDLING

Now add the capability for the GUI elements to control the actions of the program. The “get” methods in the `GUIelement` superclass will allow the contents of an input box or the current status of a checkbox or radio button to be determined, which is all that is needed for those classes. But `Button` objects need the capability to trigger actions in a program. This will be done by implementing a `Handler` superclass, and then one subclass of `Handler` for each type of action that needs to be triggered by a button. (In real Java, the classes are `EventListener` and `ActionListener` but those are too complex for COMP 1020. Take a look at them if you’re curious.)

Look at the supplied program `TestA3Q2.java`. It creates 3 buttons labelled Red, Green, and Blue. It contains three methods `drawRed()`, `drawGreen()`, and `drawBlue()`. Your job is to link each button to the correct method so that when a button is clicked, the corresponding method will be called.

- 4.1 Define a `Handler` class. It will contain no instance variables, have no constructor, and contain only the method `void doIt()`, which will do nothing. Its only purpose is to be a superclass, and allow polymorphism.
- 4.2 Define three subclasses of the `Handler` class: `HandlerRed`, `HandlerGreen`, and `HandlerBlue`. In each one, provide a `void doIt()` method which will call the corresponding `drawRed`, `drawGreen`, or `drawBlue` method in `TestA3Q2`. These “handlers” will provide the link between `Button` objects and the methods in the main program.
- 4.3 Modify your `Button` class as follows:
 - a. **Save a new copy of your class as `Button2.java`** and change the name of the class and the constructor. You must leave the `Button` class from Question 1 untouched or else the `TestA3Q1` program won’t run and you’ll lose a *lot* of marks for Question 1.
 - b. Add a new instance variable of type `Handler` that will store a reference to the `Handler` that should be used when the button is clicked. Add another parameter to the constructor (at the end) to set this variable.
 - c. Modify the `handleClick` method so that it uses the `doIt()` method of this `Handler` when the button is clicked. This should always be done after the mouse is released. Allow for the possibility that the `Handler` variable may have been set to `null`, in which case nothing should be done.
- 4.4 Modify `TestA3Q2` so that it uses `Button2` objects and not `Button` objects. Also, every time a `Button2` is created, a new `Handler` object of the correct subclass must be created and passed

to its constructor. Run the program. It should now work correctly, drawing a big coloured circle in response to button clicks.

4.5 HAND IN: Your files `Button2.java`, `Handler.java`, `HandleBlue.java`, `HandleRed.java`, and `HandleGreen.java`, as well as your *modified* version of `TestA3Q2.java`.

MARKING: The markers will run your code for both questions and a lot of marks will be based on this (probably at least half of them). The markers will 1) Place all of the files that you submitted into one folder, 2) Add a fresh copy of `TestA3Q1`, `Checkbox`, `GUIelement`, `Utilities`, and `StdDraw`, 3) Compile everything, 4) Run `TestA3Q1` and `TestA3Q2`. If anything goes wrong with a question, you will get 0 for this part of the assignment. The marker will not fix problems with the way you submitted the files. Make sure you do it correctly. Make sure these actions work.