how to interact with button presses/ActionListener

every component that you want to interact with has to have some kind of listener. for JButtons', JMenuItems', etc. you need to use an action listener. the action listener is added after the component is created, and can be any string value but usually is initalized to "this". When an ActionListener is added using "this", the objects 'ActionCommand' becomes the name of the objects text.

The ActionCommand's name is the last set text of the object, so if the object changes text, the ActionCommand will be different.

```
JButton button1 = new JButton("Button Name");
button1.addActionListener(this);


public void actionPerformed(ActionEvent ae) {// this has to be added if the class implements ActionListener
        switch (ae.getActionCommand()) {//ae.getActionCommand() gives the ActionCommand as a string, we use that string to branch to different actions in this logic tree
            case "Button Name":
            //do thing
                break;
            case "Button 2 Name":
            // do other thing
            default:
                break;
        }
    }
```

how to exit the program

with dialog

```
public void actionPerformed(ActionEvent ae){
    switch (ae.getActionCommand()){
        case "Exit":
                int exitDecision = JOptionPane.showConfirmDialog(
                        frame,
                        "Would you like to exit the program?",
                        "Exit?",
                        JOptionPane.YES_NO_OPTION);
                if (exitDecision == JOptionPane.YES_OPTION) {
                    System.exit(0);
                }
                break;
    }
}


//using the window controlls
frame.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent evt) {
        int resp = JOptionPane.showConfirmDialog(frame, "Are you sure you want to
exit?", "Exit?", JOptionPane.YES_NO_OPTION);
        if (resp == JOptionPane.YES_OPTION) {
            frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        } else {
            frame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        }
    }
});
```

without a dialog

```
//for an actionlistener named exit
case "Exit":
    System.exit(0);
    break;


frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
```

how to use a menubar

adding the JMenuBar to the frame

```
        JMenuBar menubar = new JMenuBar();
        frame.setJMenuBar(menubar);
```

adding JMenu's to a JMenubar

```
JMenu open = new JMenu("File");
open.setMnemonic('f'); //JMenu mnemonics can't be set in the constructor, so we have to use a method
menubar.add(open);
```

adding JMenuItem's to a JMenu

JMenuItems are basically just buttons, they have the same syntax, but you add them to JMenu's and JMenuBar's.

```
JMenuItem exit = new JMenuItem("Exit", 'e');
exit.addActionListener(this);
exit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_W, ActionEvent.CTRL_MASK);
```

add shortcuts

adding `Alt +` shortcuts aka mnemonics

`Alt +` shortcuts all need with a character `(char)` or an integer `(int)` that corresponds to a certain character in the alphabet. You're not supposed to use raw characters and integers and instead are supposed to use AWT's `KeyEvent.VK_` , which then converts the selected character into an integer constant, and also has the added benefit of recognizing more characters for internationalization between keyboards. You have to import the java.awt.event library for those.

I use the raw character because it's more readable and debug able though.

```
//some objects, like JButtons' and JMenuItems', let you add the alt shortcut in the object declaration
JButton open = new JButton("Open", 'o');

JMenuItem open = new JButton("Open", 'o');

//some components, like JMenus', you have to add the mnemoic through a method after the declaration because they don't have a constructor that supports adding mnemonics

JMenu file = new JMenu("File...")
file.setMnemonic('f');
```

adding other key combinations aka `CTRL +` / 'Accelerators'

Only JMenuItems' can have accelerators.

```
open.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, ActionEvent.CTRL_MASK));
```

Show a dialog

It's pretty simple

```
JOptionPane.showMessageDialog(frame,"Copyright (c) 2023 Nathan Brown","Rolodex version 0.1", JOptionPane.DEFAULT_OPTION, icon);
```

The first field is where the dialog shows up. If you set it to `null` it ignores the frames focus and just puts it on your screen, usually centered. 2nd field is a string of text inside the dialog, 3rd is the title in a string, 4th is an integer telling the method what kind of options it should return (`showMessageDialog` is void however), and the 5th and final field is an icon.

All dialogs are static, so you can't have multiple of the same type.

`showMessageDialog` is a void method but there are other types of dialog that you can set equal to variables and the program will pause until the dialog is acted on, setting the variable.

There are other types of dialogs that have non void types and can be used to get different kinds of input.

types of dialog methods for showing modal dialogs

method name	return type	constructor	comments
`showMessageDialog`	void	`showMessageDialog(Component, Object, String, int, Icon)`	The int tells wheter or not it's a `Warning` or `Message` or `Error`, etc.
`showOptionDialog`	int	`showOptionDialog(Component, Object, String, int, int, Icon, Object[], Object)`	use this for saving or discarding, things that are choices but aren't yes/no or confirm/no. The `Object[]` and `Object` at the end are all the choices and the initially selected choice (usually strings)
`showConfirmDialog`	int	`showConfirmDialog(Component, Object, String, int, int, Icon)`	use this for closing a program, confirming deletes, etc.
`showInputDialog`	String	`showInputDialog(Componenet, Object, String, int, Icon, Object[], Object)`	`Object[]` is a list of choices the user is restricted to. If `Object[]` is null, there are no choices. `Object` at the end is the initial contents of the box.

showing non-modal (inside the frame) dialogs

you can use non-modal dialogs (dialogs that are inside of a frame) by using `showInternalXxxDialog` , including `InternalInput` to get a string back, or a simple message with `InternalMessage` . More details from Oracle **here.**