GUI Programming with Java

Menus and Dialogs

- We will look at…
  - Menus in SWING
  - Dialogs in SWING

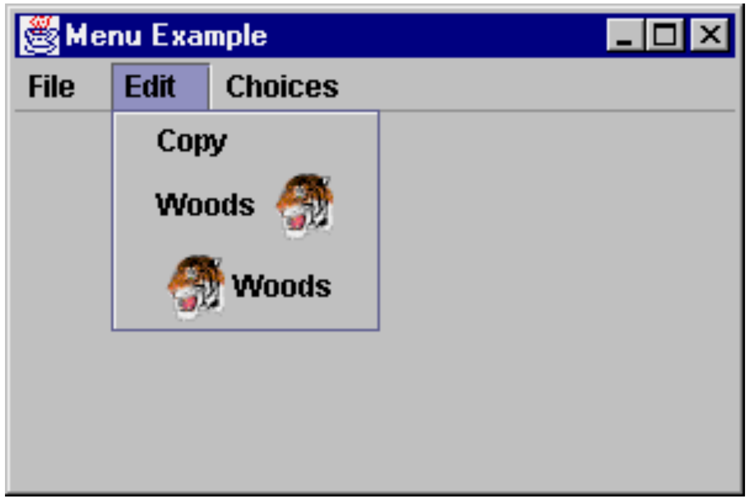
MENUS

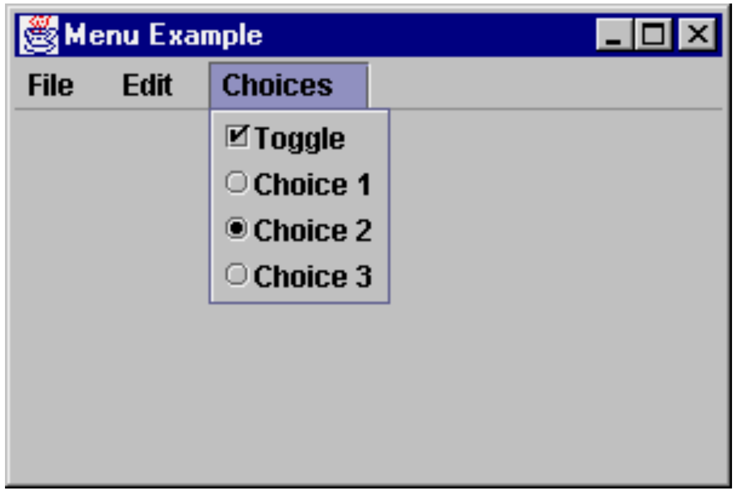
What is a Menu

- Menus are integral parts of GUI’s.
- Menus make selection easier and are widely used in window applications
- They allow the user to perform actions without unnecessarily cluttering up the graphical user interface.
- In SWING we can only apply menus to JFrame or to JApplet (both support the setJMenuBar method).

Menu Classes

- Java provides five classes to implement menus:
  - JMenuBar,
  - JMenu,
  - JMenuItem,
  - JCheckBoxMenuItem,
  - JRadioButtonMenuItem





What is a Menu Bar?

- A JFrame or JApplet can hold a *menu bar* to which the *pull-down menus* are attached.
- Menus consist of *menu items* that the user can select (or toggle on or off).
- Menu bars can be viewed as a structure to support menus
- A menu bar holds menus; the menu bar can only be added to a frame.

Adding a Menu Bar

- Following is the code to create and add a **JMenuBar** to a frame:

```
JFrame f = new JFrame();
f.setSize(300, 200);
f.setVisible(true);
JMenuBar mb = new JMenuBar();
f.setJMenuBar(mb);
```

- The Menu Bar has no menu's on it at this stage so wont really be visible.

- For an alternative way of doing this please refer to sample 1 (MenuBar.JAVA) in the sample 1 folder.

Creating Menus

- We attach menus onto a JMenuBar.
- Use the following constructor to create a menu:
  - public JMenu(String myMenuItemName)
- The following code creates two menus, File and Help, and adds them to the JMenuBar mb:

```
JMenu fileMenu = new JMenu("File", false);
JMenu helpMenu = new JMenu("Help", true);
mb.add(fileMenu);
mb.add(helpMenu);
```

- The menus will not be seen until they are added to the menu bar. For an example please refer to sample 2 (menuBar2.java) in the sample 2 folder.

Creating Menu Items

- The following code adds menu items and item separators in menu fileMenu

```
fileMenu.add(new JMenuItem("new"));
fileMenu.add(new JMenuItem("open"));
fileMenu.addSeparator();
fileMenu.add(new JMenuItem("print"));
fileMenu.add(new JMenuItem("exit"));
```

- For an example please refer to sample 3 (menuBar3.java) in the sample 3 folder.

Creating Sub Menu Items

- You can add submenus into menu items.
- The following code adds the submenus “**Unix**,” “**NT**,” and “**Win95**” into the menu item “**Software**.”

```
JMenu softwareHelpSubMenu = new JMenu("Software");
JMenu hardwareHelpSubMenu = new JMenu("Hardware");
helpMenu.add(softwareHelpSubMenu);
helpMenu.add(hardwareHelpSubMenu);

softwareHelpSubMenu.add(new JMenuItem("Unix"));
softwareHelpSubMenu.add(new JMenuItem("NT"));
softwareHelpSubMenu.add(new JMenuItem("Win95"));
```

- For an example please refer to sample 4 (menuBar4.java) in the sample 4 folder.

Create Checkbox menu items

- You can also add a JCheckBoxMenuItem to a JMenu.
- JCheckBoxMenuItem is a subclass of JMenuItem that adds a Boolean state to the JMenuItem, and displays a check when its state is true.
- You can click the menu item to turn it on and off.
- The statement following adds the checkbox menu item Check it.

```
helpMenu.add(new JCheckBoxMenuItem(“Check it”));
```

- For an example please refer to sample 5 (menuBar5.java) in the sample 5 folder.

Note: Some might consider checkboxes on menus a little outdated. They are considered by some to be bad examples of interface design (HCI)

Add Images to items

- You can add images to menu items (JMenuItem), menu checkboxes (JCheckBoxItem) and menu radio buttons (JRadioButtonMenuItem)

- You can add icons to items using the following code.

```
JMenuItem jmiNew, JmiOpen;
fileMenu.add(jmiNew = new JMenuItem(“New”));
fileMenu.add(jmiOpen = new JMenuItem(“Open”));
jmiNew.setIcon(new ImageIcon(“images/new.gif”));
jmiOpen.setIcon(new ImageIcon(“images/open.gif”));
```

- For an example please refer to sample 6 (menuBar6.java) in the sample 6 folder.

Add Images to items

- You can add images to menu items (JMenuItem), menu checkboxes (JCheckBoxItem) and menu radio buttons (JRadioButtonMenuItem)

- You can add icons to items using the following code.

```
JMenuItem jmiNew, JmiOpen;
fileMenu.add(jmiNew = new JMenuItem(“New”));
fileMenu.add(jmiOpen = new JMenuItem(“Open”));
jmiNew.setIcon(new ImageIcon(“images/new.gif”));
jmiOpen.setIcon(new ImageIcon(“images/open.gif”));
```

- For an example please refer to sample 6 (menuBar6.java) in the sample 6 folder.

Set Keyboard Mnemonics

- Setting a keyboard mnemonic for a menu item allows you to access that menu item by pressing the ALT key and the mnemonic key.
- We can add mnemonic keys to menus and menu items (including checkbox items etc).
- To add a mnemonic key to an item we use the following code
  - item.setMnemonic(key)
  - Example: helpMenu. setMnemonic('H');
- For an example please refer to sample 7 (menuBar7.java) in the sample 7 folder.

Set Keyboard Accelerators

- One problem with keyboard mnemonics is that they only let you select menu items from the currently open menu.

- Key Accelerators however, let you select a menu items directly by pressing the CTRL key and the acclerator key. For example by using the following code you can attach the accelerator key CTRL+O to the open menu item

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

- The setAccelerator method takes an object KeyStroke. The static method getKeyStroke in the KeyStroke class creates an instance of the keystroke.

- VK_O is a constant representing the O key and CTRL_MASK is a constant indicating that the CTRL key is associated with the keystroke.

- For an example please refer to sample 8 (menuBar8.java) in the sample 8 folder.

Adding Event Handling

- Event handling for menu items is pretty straightforward.
- Menu items generate ActionEvent objects. Your program must implement actionPerformed handler to respond to the menu selection.
- For an example of event handling for menus please refer to sample 9 (MenuBar9.java) in the sample 9 folder.

Enabling or Disabling Menu Items

- One way of protecting users from making mistakes is to disable menu items when they are not appropriate.
- To achieve this we can use the setEnabled method.
- The format of the method is setEnabled(boolean)
- Example: jmiNew.setEnabled(false)
- The above line of code will disable the new file menu item.

Summary

- IN SWING we can only apply menus to JFrame or to JApplet (both support the setJMenuBar method).
- Java provides five classes to implement menus:
  - JMenuBar,
  - JMenu,
  - JMenuItem,
  - JCheckBoxMenuItem,
  - JRadioButtonMenuItem
- A JFrame or JApplet can hold a *menu bar* to which the *pull-down menus* are attached.
- A menu bar holds menus; the menu bar can only be added to a frame.

Summary(2)

- The following code adds menu items and item separators in menu fileMenu

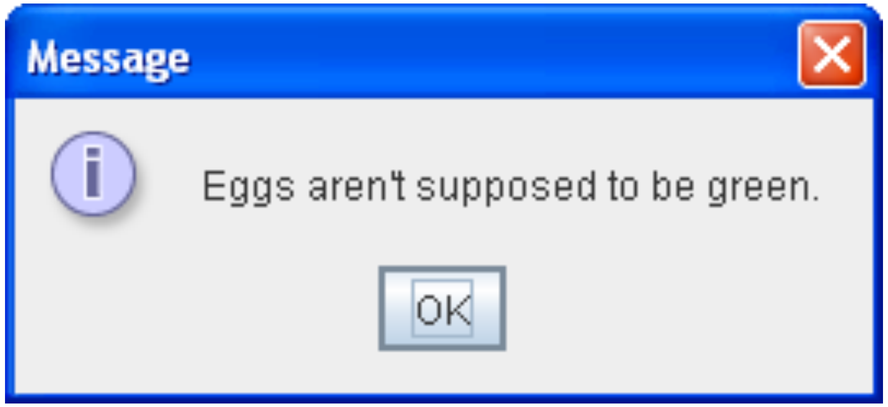
```
fileMenu.add(new JMenuItem("new"));
fileMenu.add(new JMenuItem("open"));
fileMenu.addSeparator();
fileMenu.add(new JMenuItem("print"));
fileMenu.add(new JMenuItem("exit"));
```

- You can add submenus into menu items.
- We can also add checkboxes and radio buttons to menus (and as sub items to menu items).
- By using the setIcon method we can add an icon to a menu item.
- Keyboard accelerators and mnemonics allow us faster access to menu items.
- Event handling is achieved via the use of the actionPerformed handler.

DIALOGS

- A dialog is basically a window that is more limited than frame.
- Several classes support *dialogs*.
- The ProgressMonitor class can put up a dialog that shows the progress of an operation.
- To bring up a print dialog, you can use the Printing API.
- To create custom dialogs, use the JDialog class directly.

- The code for simple dialogs can be minimal. For example, here's an informational dialog:



Eggs aren't supposed to be green.

- Here is the code that creates and shows it:

```
JOptionPane.showMessageDialog(frame, "Eggs aren't supposed to be green.");
```

An Overview of Dialogs

- Every dialog is dependent on a frame.
- When that frame is destroyed, so are its dependent dialogs.
- When the frame is iconified, its dependent dialogs disappear from the screen.
- When the frame is deiconified, its dependent dialogs return to the screen. SWING automatically provides this behavior.

modal Dialogs

- A dialog can be *modal* (by default most are modal)
- When a modal dialog is visible, it blocks user input to all other windows in the program.
- The JDialogs that JOptionPane creates are modal.
- To create a non-modal dialog, you must use the JDialog class directly.

Dialog Examples

- Lets take a look at the DialogDemo.java example in the sample 2 – dialog folder
- This sample demonstrates many of the different types of dialogs that we can use in Java

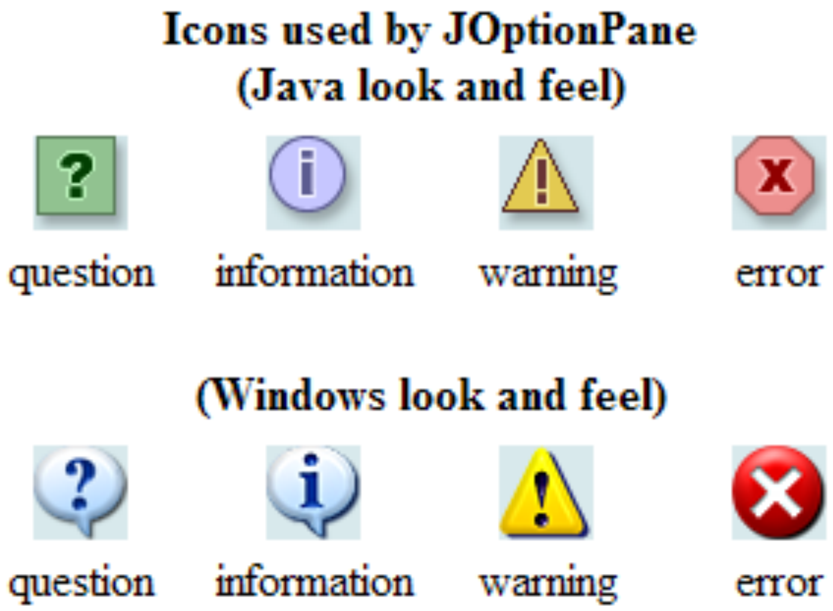
JOptionPane Features

- Using JOptionPane, you can create and customize several different kinds of dialogs.
- JOptionPane provides support for laying out standard dialogs, providing icons, specifying the dialog's title and text, and customizing the button text.
- Other features allow you to customize the components the dialog displays and specify where the dialog should appear onscreen.
- You can even specify that an option pane put itself into an internal frame (JInternalFrame) instead of a JDialog.

JOptionPane Features

- JOptionPane's icon support lets you easily specify which icon the dialog displays.
- You can use a custom icon, no icon at all, or any one of four standard JOptionPane icons
  - Question
  - Information
  - Warning
  - Error

- Each look and feel has its own versions of the four standard icons. The following figure shows the icons used in the Java look and feel.



Creating and Showing Simple Dialogs

- For most simple modal dialogs, you create and show the dialog using one of JOptionPane's show*Xxx*Dialog methods.
- If your dialog should be an internal frame, then add Internal after show — for example, showMessageDialog changes to showInternalMessageDialog.

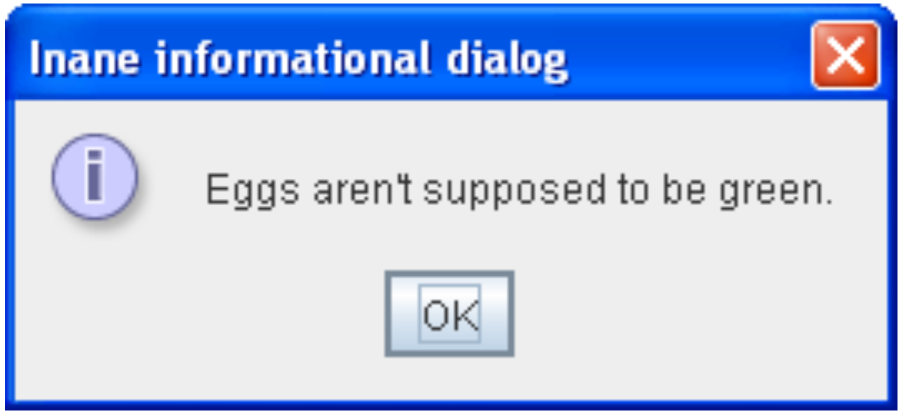
- The two most useful show*Xxx*Dialog methods are showMessageDialog and showOptionDialog

- The showMessageDialog method displays a simple, one-button dialog.

- The showOptionDialog method displays a customized dialog — it can display a variety of buttons with customized button text, and can contain a standard text message or a collection of components.

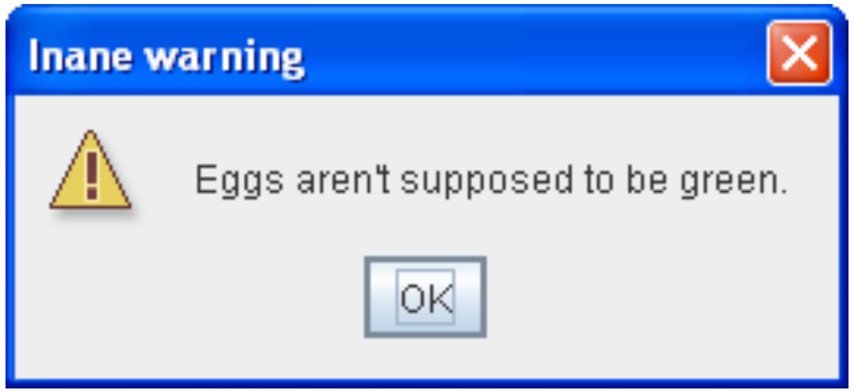
showMessageDialog

Lets take a look at some examples


Inane informational dialog

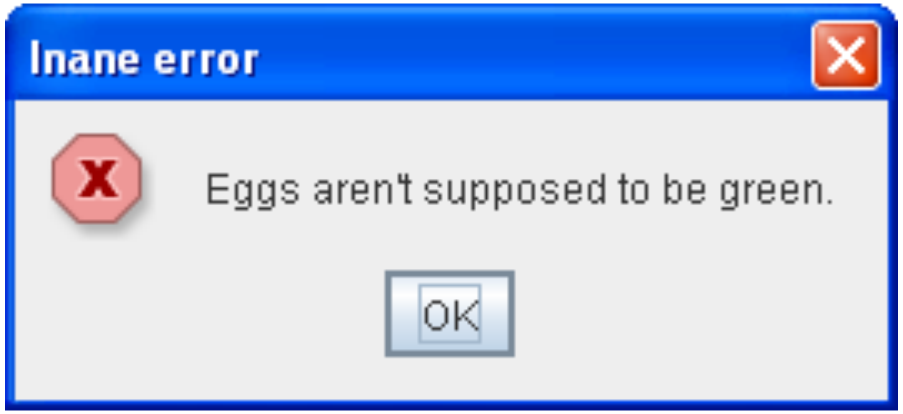
Eggs aren't supposed to be green.

```
JOptionPane.showMessageDialog(frame, "Eggs aren't supposed to be green.");
```



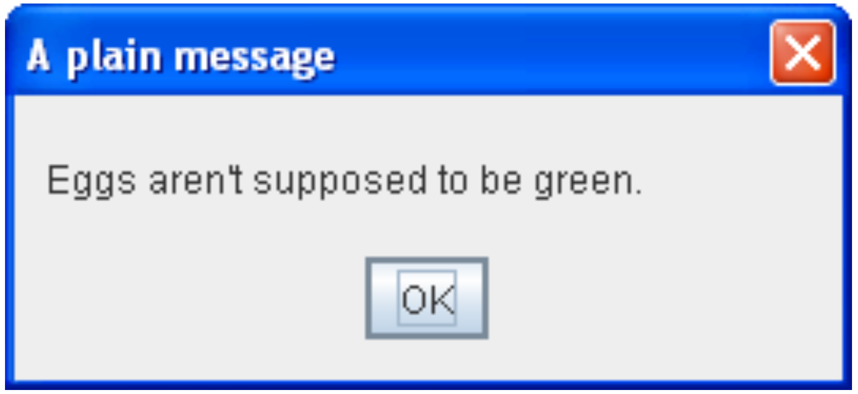
Eggs aren't supposed to be green.

```
JOptionPane.showMessageDialog(frame, "Eggs aren't supposed to be green.",
"Inane warning", JOptionPane.WARNING_MESSAGE);
```



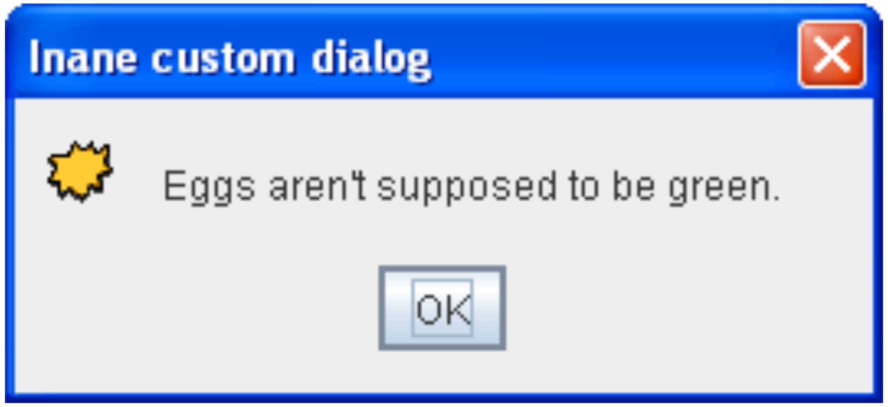
Eggs aren't supposed to be green.

```
JOptionPane.showMessageDialog(frame, "Eggs aren't supposed to be green.",
"Inane error", JOptionPane.ERROR_MESSAGE);
```



Eggs aren't supposed to be green.

```
JOptionPane.showMessageDialog(frame, "Eggs aren't supposed to be green.",
        "A plain message", JOptionPane.PLAIN_MESSAGE);
```

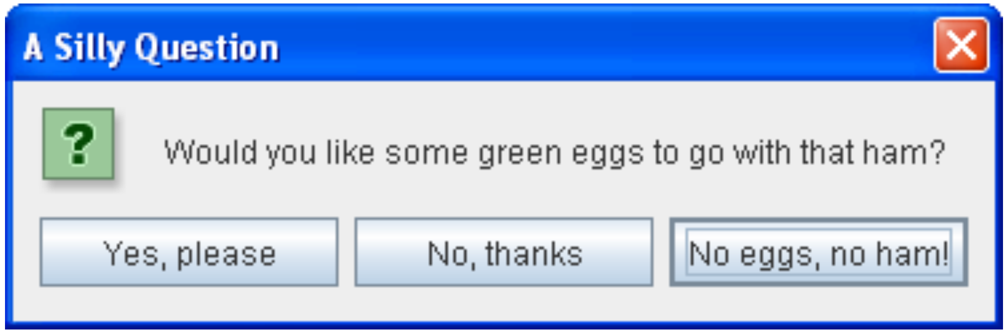


Eggs aren't supposed to be green.

```
JOptionPane.showMessageDialog(frame, "Eggs aren't supposed to be green.",
"Inane custom dialog", JOptionPane.INFORMATION_MESSAGE, icon);
```

showOptionDialog and showConfirmDialog

Lets take a look at some examples

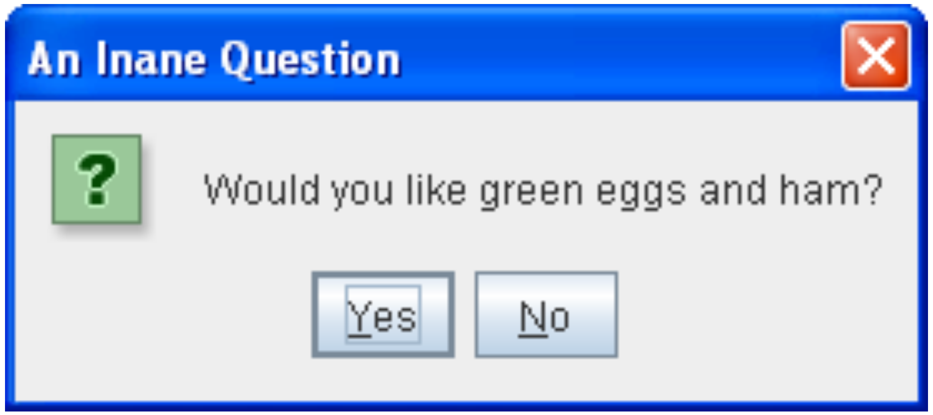


Would you like some green eggs to go with that ham?

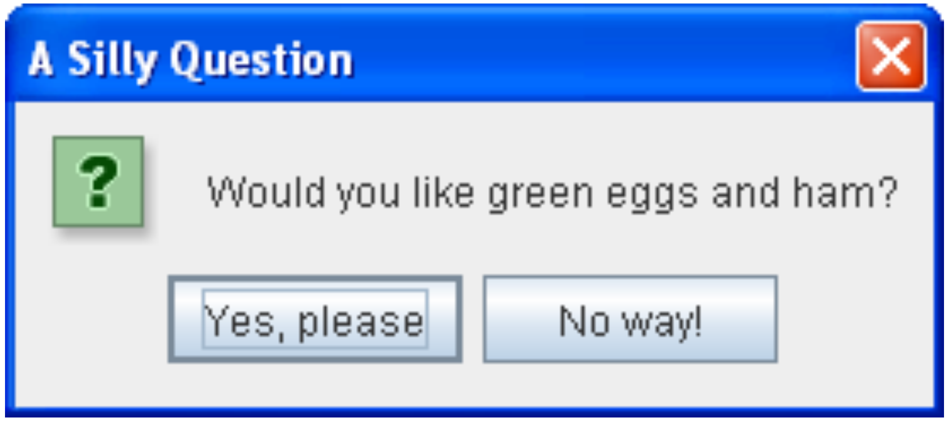
```
//Custom button text

Object[] options = {"Yes, please", "No, thanks", "No eggs, no
ham!"};

int n = JOptionPane.showOptionDialog(frame, "Would you like
some green eggs to go " + "with that ham?", "A Silly Question",
JOptionPane.YES_NO_CANCEL_OPTION,
JOptionPane.QUESTION_MESSAGE, null, options, options[2]);
```



```
//default icon, custom title
int n = JOptionPane.showConfirmDialog( frame, "Would you like
green eggs and ham?", "An Inane Question",
JOptionPane.YES_NO_OPTION);
```



```
Object[] options = {"Yes, please", "No way!"};

int n = JOptionPane.showOptionDialog(frame, "Would you like green eggs
and ham?", "A Silly Question", JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE, null, //don't use a custom Icon
options, //the titles of buttons options[0]); //default button title
```

Return Values

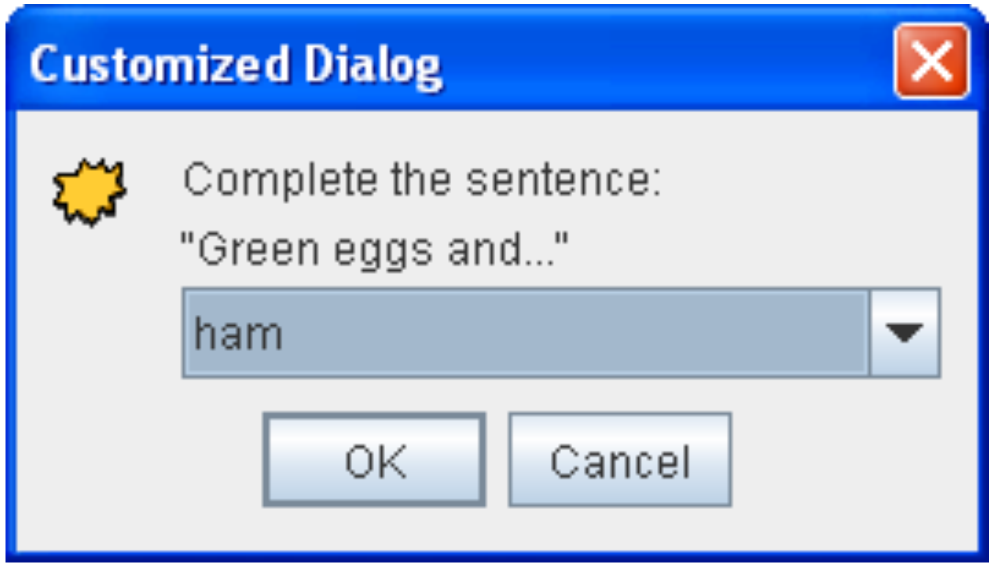
- As the previous code snippets showed, the showMessageDialog, showConfirmDialog, and showOptionDialog methods return an integer indicating the user's choice.
- The values for this integer are YES_OPTION, NO_OPTION, CANCEL_OPTION, OK_OPTION, and CLOSED_OPTION.
- Except for CLOSED_OPTION, each option corresponds to the button the user pressed. When CLOSED_OPTION is returned, it indicates that the user closed the dialog window explicitly, rather than by choosing a button inside the option pane.
- Even if you change the strings that the standard dialog buttons display, the return value is still one of the pre-defined integers. For example, a YES_NO_OPTION dialog always returns one of the following values: YES_OPTION, NO_OPTION, or CLOSED_OPTION.

showInputDialog

Lets take a look at some examples

- The only form of show*Xxx*Dialog that doesn't return an integer is showInputDialog, which returns an Object instead.

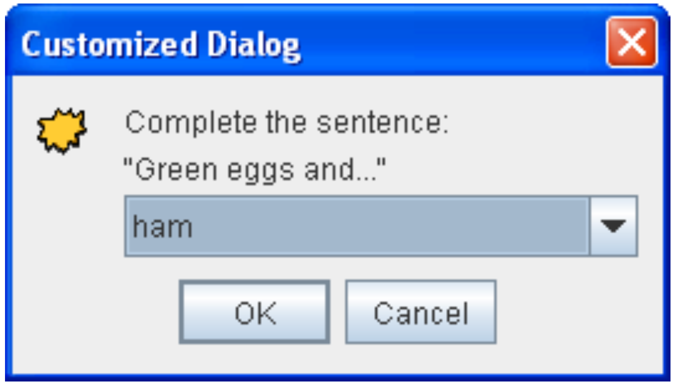
- This Object is generally a String reflecting the user's choice.



Complete the sentence:
"Green eggs and..."
ham

```
Object[] possibilities = {"ham", "spam", "yam"};
String s = (String)JOptionPane.showInputDialog(
                    frame,
                    "Complete the sentence:\n"
                    + "\"Green eggs and...\"",
                    "Customized Dialog",
                    JOptionPane.PLAIN_MESSAGE,
                    icon,
                    possibilities,
                    "ham");

//If a string was returned, say so.
if ((s != null) && (s.length() > 0)) {
          setLabel("Green eggs and... " + s + "!"); return;
}
//If you're here, the return value was null/empty.
setLabel("Come on, finish the sentence!");
```


Customized Dialog

Complete the sentence:

"Green eggs and..."