

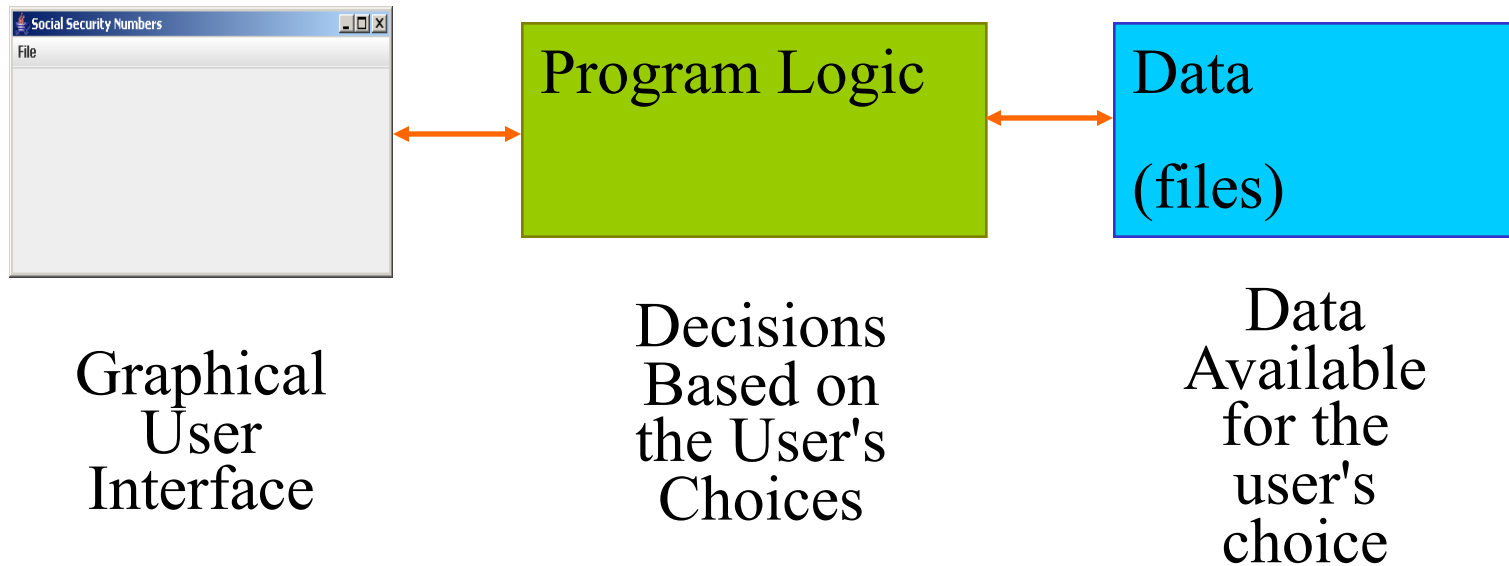
# GUIs: Event-Driven Programming

Complete JFrames

# JFrames

- Complete Window objects
- They don't do anything until told to
- An *event*, such as a menu choice, signals the JFrame to respond.
- This is called *Event-Driven Programming*

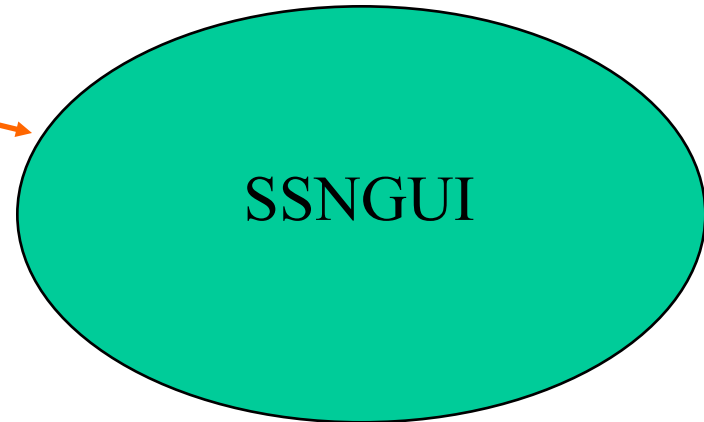
# "Three Tier Architecture"



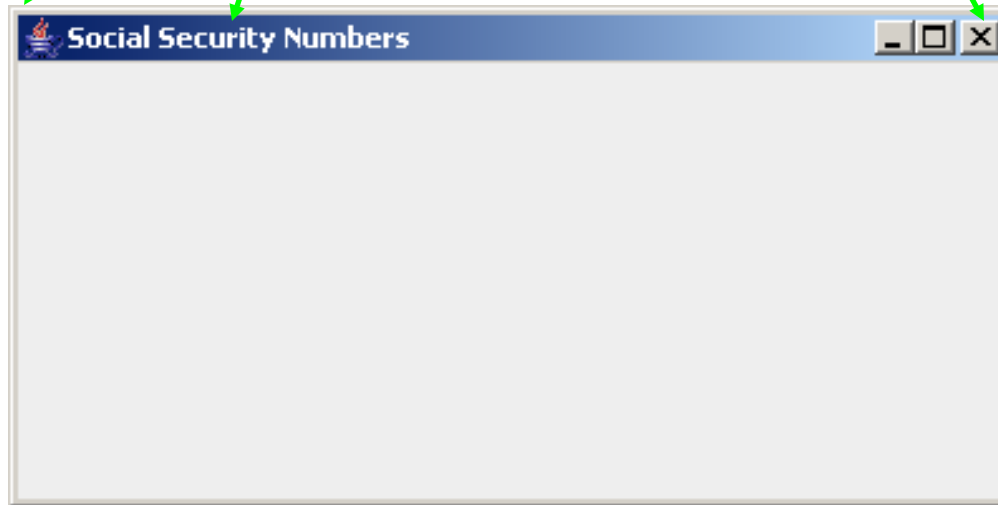
All the main program needs to do is instantiate the GUI

```
public class SSNMain {  
    static SSNGUI ssnGUI;  
    public static void main(String[] args) {  
        ssnGUI = new SSNGUI("My SSN GUI", 400,200);  
    }  
}
```

ssnGUI

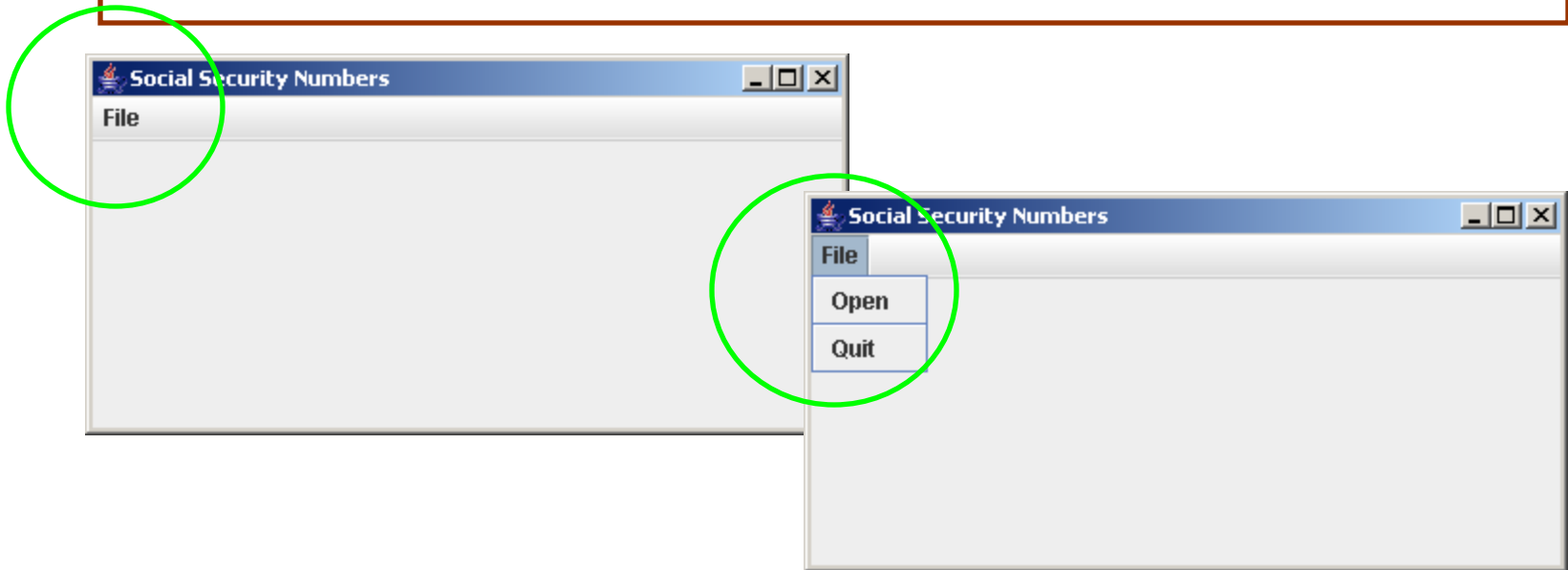


```
public SSNGUI(String title, int height, int width) {  
    setTitle(title);  
    setSize(height,width);  
    setLocation (400,200);  
    setDefaultCloseOperation(EXIT_ON_CLOSE);  
    setVisible(true);  
} //SSNGUI
```



## Add a menu...

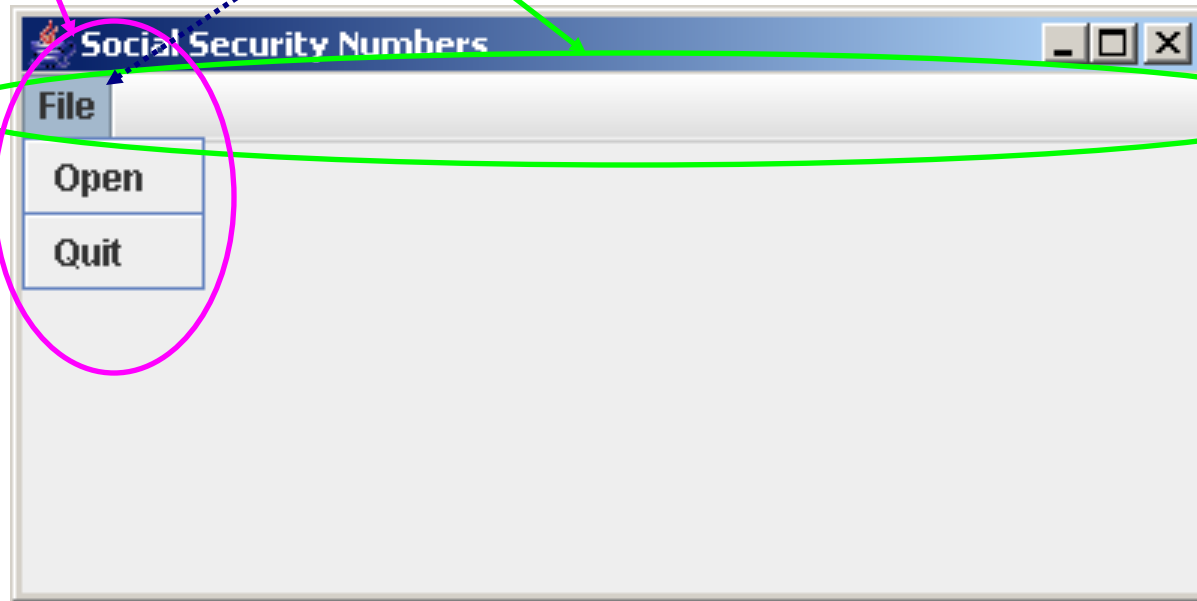
```
public SSNGUI(String title, int height, int width) {  
    setTitle(title);  
    setSize(height,width);  
    setTitle("Social Security Numbers");  
    setLocation(400,200);  
    createMenu() ;  
    setDefaultCloseOperation(EXIT_ON_CLOSE);  
    setVisible(true);  
}
```



## The basics of creating drop-down menus...

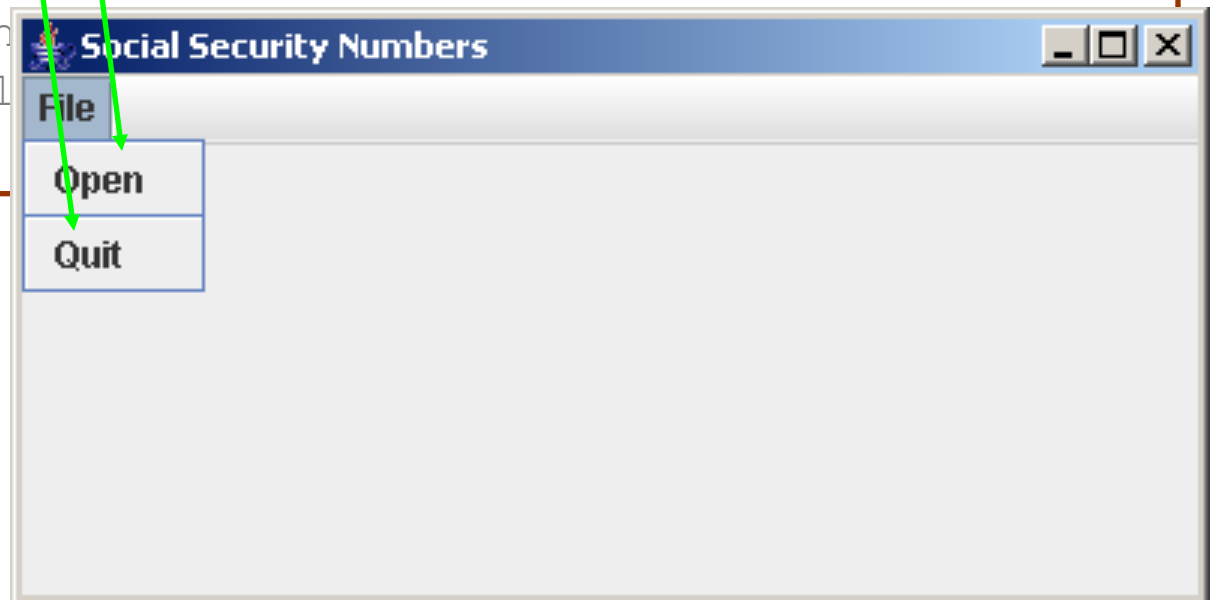
```
private void createMenu( ) {  
    JMenuBar menuBar = new JMenuBar();  
    JMenu fileMenu = new JMenu("File");  
    JMenuItem item;  
    FileMenuHandler fmh = new FileMenuHandler(this);  
    item = new JMenuItem("Open");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    fileMenu.addSeparator();  
    item = new JMenuItem("Quit");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    setJMenuBar(menuBar);  
    menuBar.add(fileMenu);  
} //createMenu
```

```
private void createMenu( ) {  
    JMenuBar menuBar = new JMenuBar();  
    JMenu fileMenu = new JMenu("File");  
    JMenuItem item;  
}  
//createMenu
```





```
private void createMenu( ) {  
    JMenuBar menuBar = new JMenuBar();  
    JMenu fileMenu = new JMenu("File");  
    JMenuItem item;  
    FileMenuHandler fmh = new FileMenuHandler(this);  
    item = new JMenuItem("Open");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    fileMenu.addSeparator();  
    item = new JMenuItem("Quit");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    setJMenuBar( menuBar );  
    menuBar.add( fileMenu );  
} //createMenu
```



```
private void createMenu( ) {  
    JMenuBar menuBar = new JMenuBar();  
    JMenu fileMenu = new JMenu("File");  
    JMenuItem item;  
    FileMenuHandler fmh = new FileMenuHandler(this);  
    item = new JMenuItem("Open");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    fileMenu.addSeparator();  
    item = new JMenuItem("Quit");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    menuBar.add(fileMenu);  
    setJMenuBar(menuBar);  
} //createMenu
```



Put the fileMenu in  
the menuBar



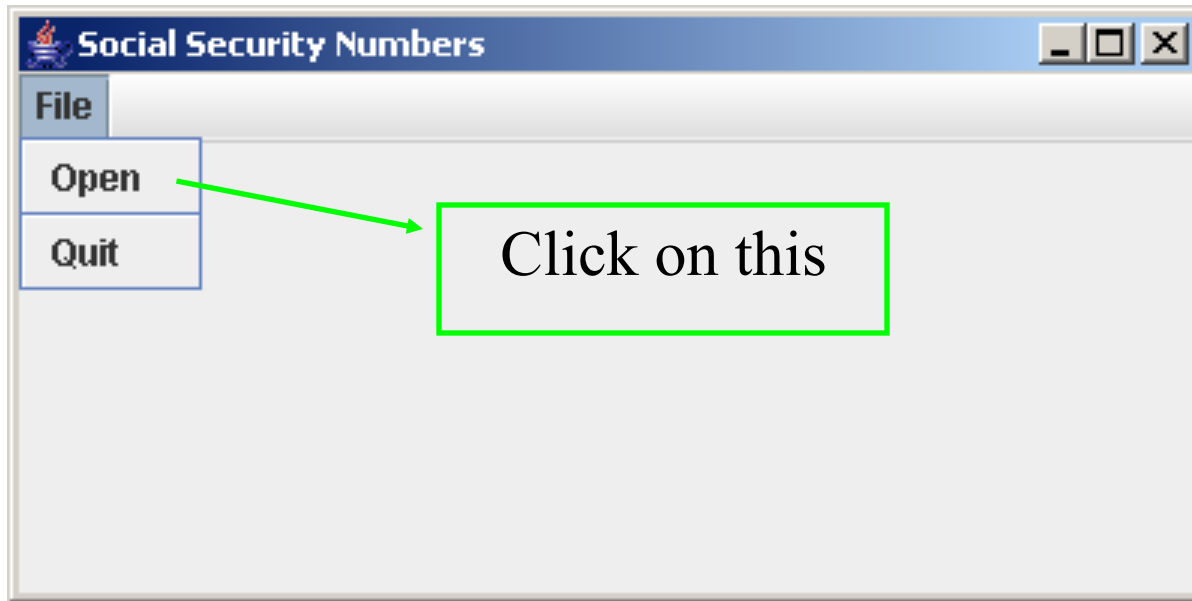
Put the menuBar in  
the JFrame

Now we handle the *events*

```
private void createMenu( ) {  
    JMenuBar menuBar = new JMenuBar();  
    JMenu fileMenu = new JMenu("File");  
    JMenuItem item;  
    FileMenuHandler fmh = new FileMenuHandler(this);  
    item = new JMenuItem("Open");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    fileMenu.addSeparator();  
    item = new JMenuItem("Quit");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    menuBar.add(fileMenu);  
    setJMenuBar(menuBar);  
} //createMenu
```

An *event* is something that happens while the program is running

- The user clicks on the X to close the window
- The user chooses an item from a menu



# *Events* need to be handled

- An *Event Handler* is a method that is automatically called when an event, such as choosing a menu item, occurs.
- An *Event Handler* can handle more than one event, but each event needs a handler (or nothing will happen)
- Event Handlers* are written in a class that implements an interface called *ActionListener*

# What is an *Interface*?

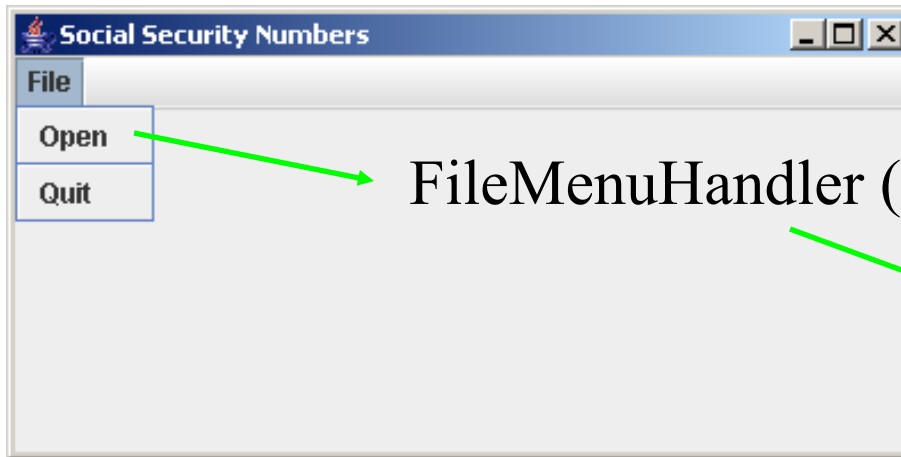
- An *Interface* is a collection of method headings only (not bodies).
- Interfaces are *implemented* by a Java class
- If an interface is implemented, **all** methods specified in the interface must be provided by that class
- An interface, if implemented, guarantees that all methods will be defined.

```
public interface X {  
    public int y (int z);  
    public void q();  
}
```

```
public class A implements X {  
    public int y (int z) { return z+1 }  
    public void q() { }  
}
```

# Interface *ActionListener*

- Contains a method heading *actionPerformed(ActionEvent)*
- The *actionPerformed* method is called when an event happens.
- Each event needs to be *registered* with some ActionListener



FileMenuHandler (implements ActionListener)

actionPerformed called

```
private void createMenu( ) {  
    JMenuBar menuBar = new JMenuBar();  
    JMenu fileMenu = new JMenu("File");  
    JMenuItem item;  
    FileMenuHandler fmh = new FileMenuHandler(this);  
    item = new JMenuItem("Open");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    fileMenu.addSeparator();  
    item = new JMenuItem("Quit");  
    item.addActionListener( fmh );  
    fileMenu.add( item );  
    menuBar.add(fileMenu);  
    setJMenuBar(menuBar);  
} //createMenu
```

Here, clicking on *open* or on *quit* will call the `actionPerformed` method of the `FileMenuHandler` class



...

```
FileMenuHandler fmh = new FileMenuHandler(this);
```

```
item = new JMenuItem("Open");
```

```
item.addActionListener( fmh);
```

```
item = new JMenuItem("Quit");
```

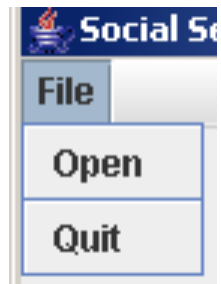
```
item.addActionListener( fmh );
```

...

fmh



item



...

```
FileMenuHandler fmh = new FileMenuHandler(this);
```

```
item = new JMenuItem("Open");
```

```
item.addActionListener( fmh);
```

```
item = new JMenuItem("Quit");
```

```
item.addActionListener( fmh );
```

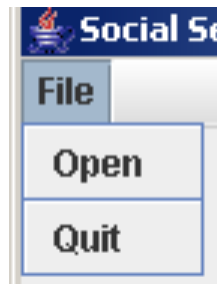
...

fmh

item

## *FileMenuHandler*

```
public FileMenuHandler (JFrame jf) {  
    jframe = jf;  
}
```



...

```
FileMenuHandler fmh = new FileMenuHandler(this);
```

```
item = new JMenuItem("Open");
```

```
item.addActionListener( fmh);
```

```
item = new JMenuItem("Quit");
```

```
item.addActionListener( fmh );
```

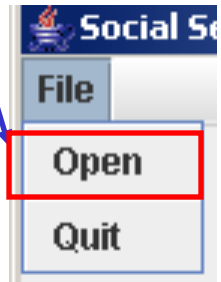
...

fmh

item

## *FileMenuHandler*

```
public FileMenuHandler (JFrame jf) {  
    jframe = jf;  
}
```



...

```
FileMenuHandler fmh = new FileMenuHandler(this);
```

```
item = new JMenuItem("Open");
```

```
item.addActionListener( fmh );
```

```
item = new JMenuItem("Quit");
```

```
item.addActionListener( fmh );
```

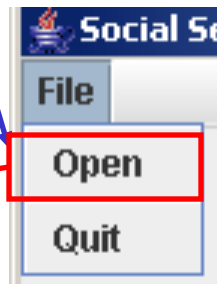
...

fmh

item

## *FileMenuHandler*

```
public FileMenuHandler (JFrame jf) {  
    jframe = jf;  
}
```



...

```
FileMenuHandler fmh = new FileMenuHandler(this);
```

```
item = new JMenuItem("Open");
```

```
item.addActionListener( fmh);
```

```
item = new JMenuItem("Quit");
```

```
item.addActionListener( fmh );
```

...

fmh

item

*FileMenuHandler*

```
public FileMenuHandler (JFrame jf) {  
    jframe = jf;  
}
```



...

```
FileMenuHandler fmh = new FileMenuHandler(this);
```

```
item = new JMenuItem("Open");
```

```
item.addActionListener( fmh );
```

```
item = new JMenuItem("Quit");
```

```
item.addActionListener( fmh );
```

...

fmh

item

*FileMenuHandler*

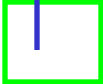
```
public FileMenuHandler (JFrame jf) {  
    jframe = jf;  
}
```



...


```
FileMenuHandler fmh = new FileMenuHandler(this);  
item = new JMenuItem("Open");  
item.addActionListener( fmh);  
item = new JMenuItem("Quit");  
item.addActionListener( fmh );  
}
```

### *FileMenuHandler*

```
public FileMenuHandler (JFrame jf) {  
    jframe = jf;   
}
```

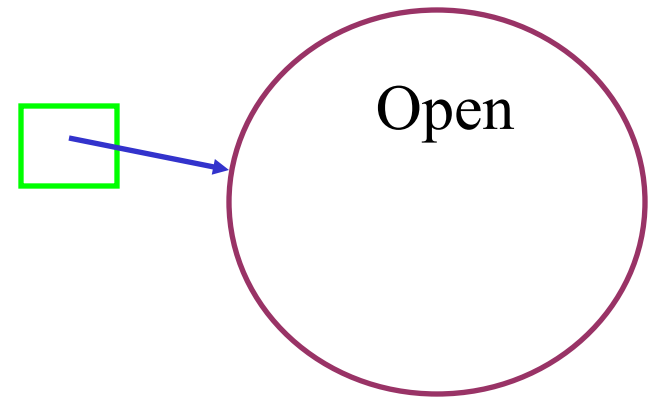


## *FileMenuHandler*

```
public FileMenuHandler (JFrame jf) {  
    jframe = jf;   
}
```




- Suppose the user clicks on "Open"
- An *ActionEvent* object is created

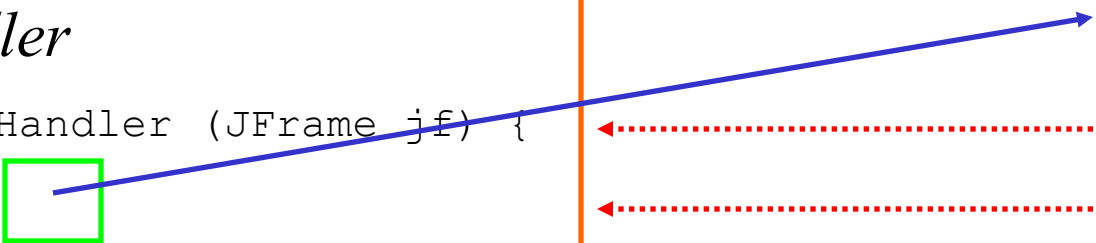



- The *actionPerformed* method of the handler that is registered with the event is called.
- The *ActionEvent* is passed to it as a parameter

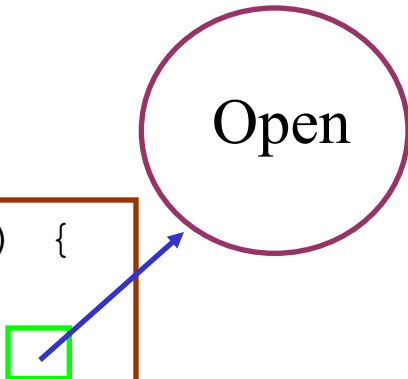


## *FileMenuHandler*

```
public FileMenuHandler (JFrame jf) {  
    JFrame = jf;   
}
```



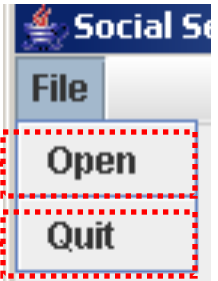
```
public void actionPerformed(ActionEvent event) {  
    String menuName;  
    menuName = event.getActionCommand(); event   
    if (menuName.equals("Open"))  
        openFile( );  
    else if (menuName.equals("Quit"))  
        System.exit(0);  
} //actionPerformed
```



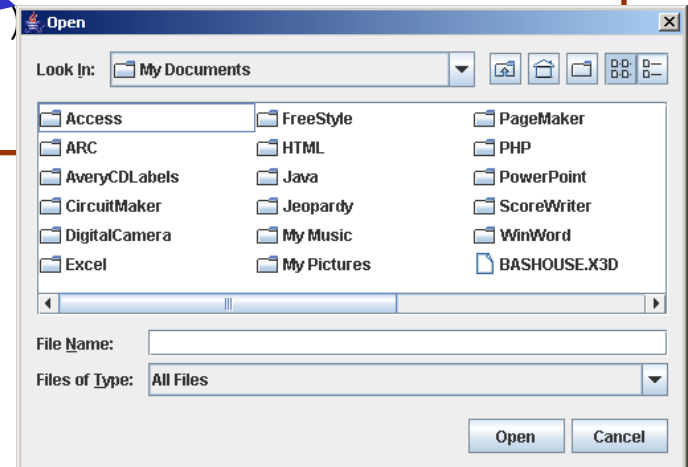
Open

## *FileMenuHandler*

```
public FileMenuHandler (JFrame jf) {  
    jframe = jf;       
}
```

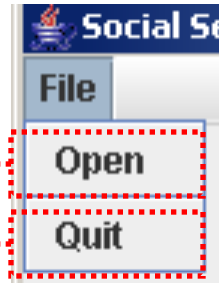


```
private void openFile( ) {  
    int status;  
  
    JFileChooser chooser = new JFileChooser( );  
    status = chooser.showOpenDialog(null);  
    readSource(chooser.getSelectedFile());  
} //openFile
```



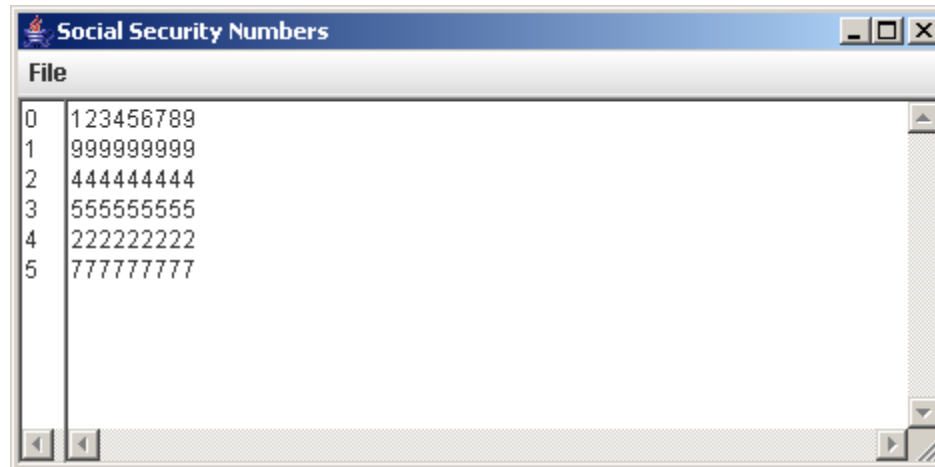
## *FileMenuHandler*

```
public FileMenuHandler (JFrame jf) {  
    jframe = jf;  
}
```



```
private void readSource(File chosenFile) {  
    String chosenFileName = chosenFile.getName();  
    TextFileInput inFile = new TextFileInput(chosenFileName);  
    String ssn;  
    int subscript = 0;  
    Container myContentPane = jframe.getContentPane();  
    JTextArea myTextArea = new JTextArea();  
    JTextArea mySubscripts = new JTextArea();  
    myContentPane.add(myTextArea, BorderLayout.EAST);  
    myContentPane.add(mySubscripts, BorderLayout.WEST);  
    ...  
}
```

```
...
ssn = inFile.readLine();
while (ssn != null) {
    mySubscripts.append(Integer.toString(subscript++)+"\n");
    myTextArea.append(ssn+"\n");
    ssn = inFile.readLine();
} //while
jframe.setVisible(true);
```



Main program starts and instantiates a GUI

GUI creates its JMenuBar, JMenu and JMenuItem

GUI registers each JMenuItem with one or more  
ActionListeners

User clicks on a JMenuItem

The actionPerformed method in the handler registered  
with that JMenuItem is called

GUI waits for another event to happen