GUI Programming, Part 2

CST 205

Review

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget

app = QApplication(sys.argv)

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.show()

ex = Example()
status = app.exec_()
sys.exit(status)
```

Review

- The core of every PyQt5 application is the `QApplication` class

- Your GUI application must have one (and only one) `QApplication` object

- `QApplication`'s `exec_()` method starts the **main loop**.
  - Sometimes called the **event loop**.

- `QWidget` class is the base class of all user interface objects.

`QMainWindow` class

- We call also create a window using the **`QMainWindow`** class.
  - The **`QMainWindow`** class provides a main application window.

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow

app = QApplication(sys.argv)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('CST 205 Main Window')

mainWin = MainWindow()
mainWin.show()
status = app.exec_()
sys.exit(status)
```

Layouts

- Qt provides three layout managers:
  - vertical (`QVBoxLayout`)
  - horizontal (`QHBoxLayout`)
  - grid (`QGridLayout`)
- Layouts can be nested
  - allows for very sophisticated layouts

Basic Layout

- Create a layout object:

```
layout = QVBoxLayout()
```

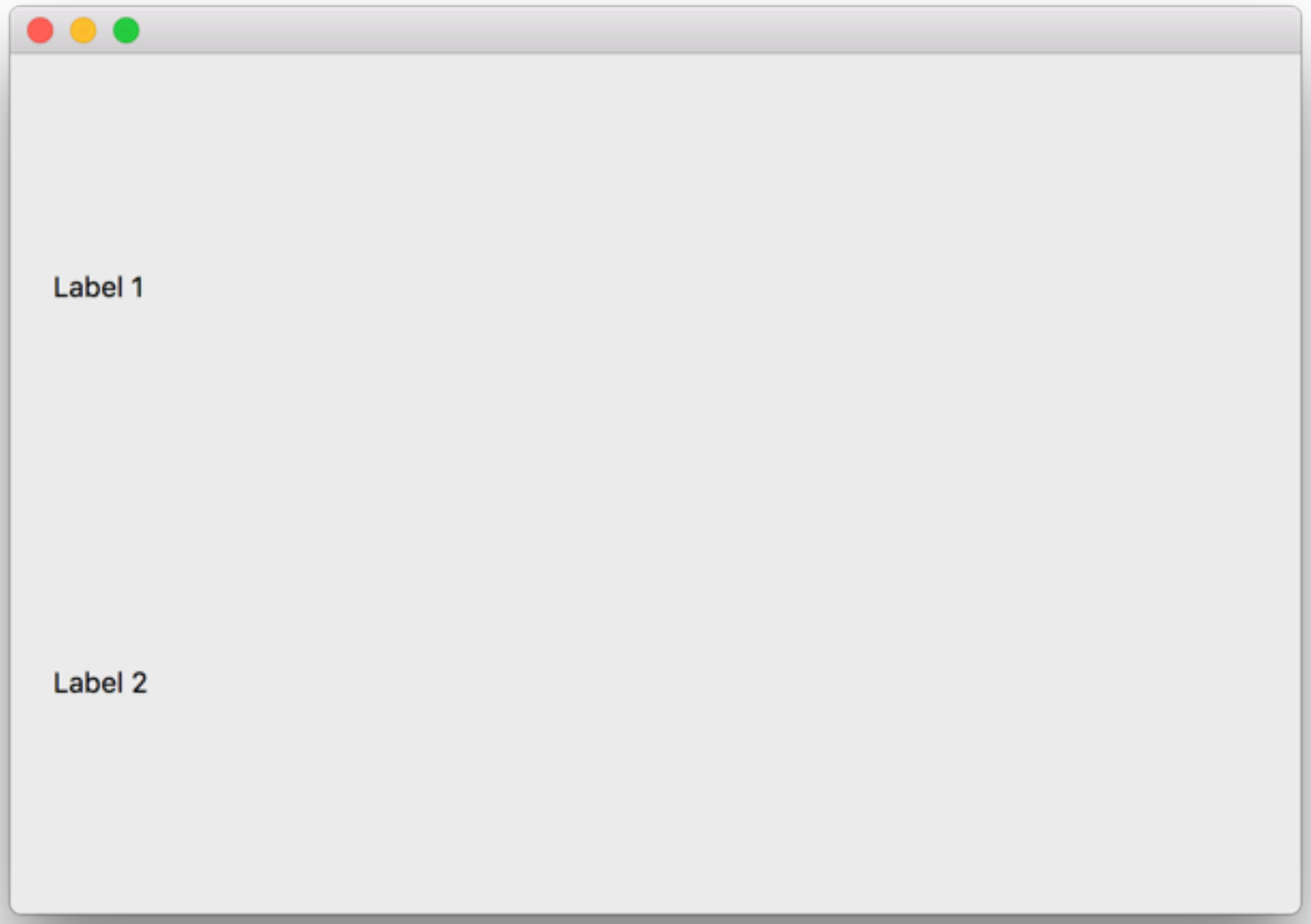
- Use the layout's `addWidget()` method to add widgets such as buttons, text fields, and even other layouts.

- Supply the layout to the window using the window's `setLayout()` method.

Layout code sample

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout

class Example(QWidget):
    def __init__(self):
        super().__init__()

        self.label1 = QLabel('Label 1', self)
        self.label2 = QLabel('Label 2', self)

        vbox = QVBoxLayout()

        vbox.addWidget(self.label1)
        vbox.addWidget(self.label2)

        self.setLayout(vbox)
        self.setGeometry(100,100,600,400)
        self.show()

app = QApplication(sys.argv)
ex = Example()
sys.exit(app.exec_())
```

Layout code result



Capturing User Interaction

- **Signal-Slot** is one of the fundamental topics of Qt.

- A *signal* is emitted when something of potential interest happens.

- If a signal is connected to a *slot*, then the slot is called when the signal is emitted.

  - If a signal isn't connected, then nothing happens.

- The event loop continuously checks if there is an event to process.

Button click example

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout, QLabel
from PyQt5.QtCore import pyqtSlot

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()

        vbox = QVBoxLayout()

        self.my_btn = QPushButton("Button 1", self)
        self.my_lbl = QLabel('Button not clicked')
        self.my_btn.clicked.connect(self.on_click)

        vbox.addWidget(self.my_btn)
        vbox.addWidget(self.my_lbl)
        self.setLayout(vbox)

    @pyqtSlot()
    def on_click(self):
        self.my_lbl.setText('Button clicked')


app = QApplication(sys.argv)
main_win = MainWindow()
main_win.show()
sys.exit(app.exec_())
```

Spinbox and Dial example

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QDial, QHBoxLayout, QSpinBox

class Form(QWidget):
    def __init__(self):
        super().__init__()

        self.dial = QDial()
        self.dial.setNotchesVisible(True)
        self.spinbox = QSpinBox()

        layout = QHBoxLayout()
        layout.addWidget(self.dial)
        layout.addWidget(self.spinbox)

        self.setLayout(layout)
        self.setWindowTitle("Signals and Slots")
        self.dial.valueChanged.connect(self.spinbox.setValue)
        self.spinbox.valueChanged.connect(self.dial.setValue)

app = QApplication(sys.argv)
ex = Form()
ex.show()
sys.exit(app.exec_())
```

Spinbox and Dial