NHERI SIMCENTER PROGRAMMING BOOTCAMP

JULY 30 THROUGH AUGUST 3, 2018, AT UC BERKELEY'S RICHMOND FIELD STATION

GUI Development

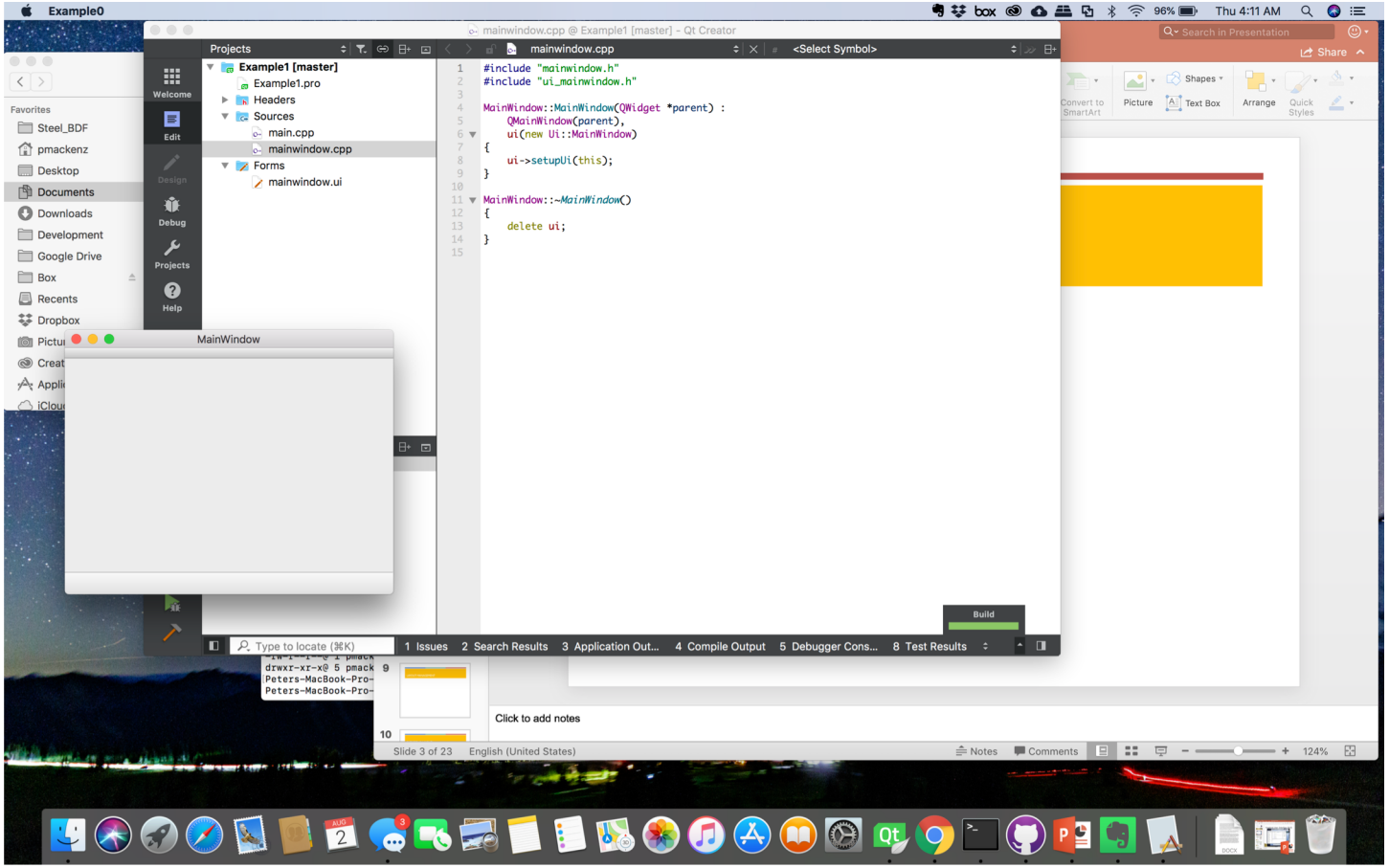
SimCenter NHERI
Center for Computational Modeling and Simulation

OUTLINE

- GUI Design Fundamentals
- The Qt Framework
  - Common Data Types/Classes
  - Building the UI
  - Layout Management
  - Signals and Slots
  - Model – View – Controller Concept
  - Helper Widgets
- Quite a few Exercise Sessions

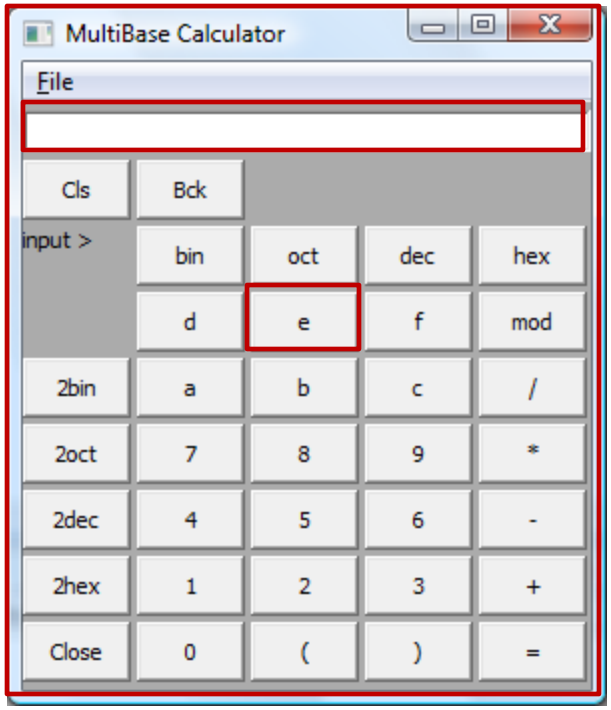
GUI FUNDAMENTALS

- What is a WINDOW?



GUI FUNDAMENTALS

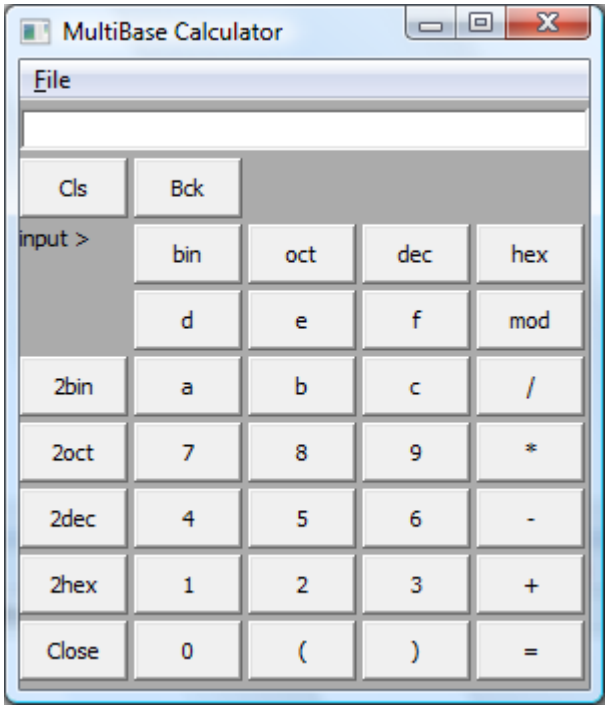
- What is a WINDOW?
  - "A rectangular area on your screen"
  - "Any rectangular area on your screen"



GUI FUNDAMENTALS

Characteristics of an Application with a GUI

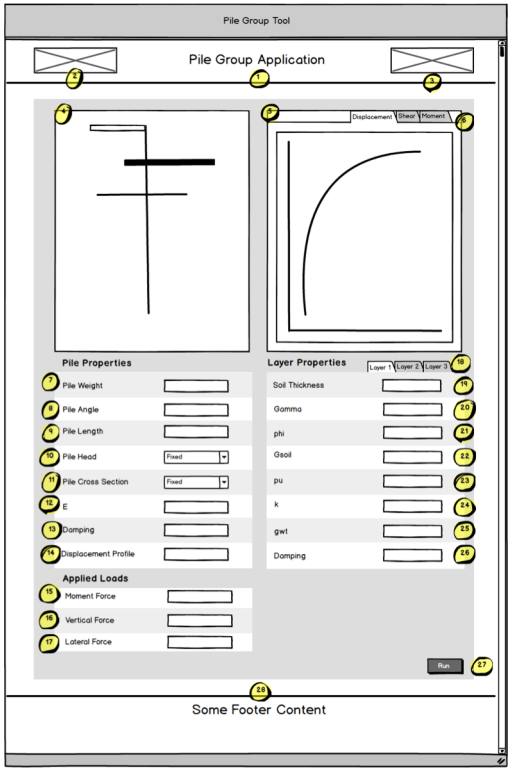
- Arbitrary sequence of execution
- May change shape/size
- May be (partially) covered
- Can be active or inactive



DESIGNING AN APPLICATION

1. CLOSE YOUR LAPTOP/WALK AWAY FROM YOUR COMPUTER !
2. Define target requirements – write them down !
   - Basic functionality
   - Available/required input
   - Desired outcome/output
3. Develop User Interface (UI)
   1. Sketch on paper/whiteboard/napkin/BART ticket/etc.
   2. Redo a few times till you like it; Draw a large sketch of the final version
   3. Identify all objects by type and functionality
   4. Play use-scenarios on paper
   5. Update your design as needed

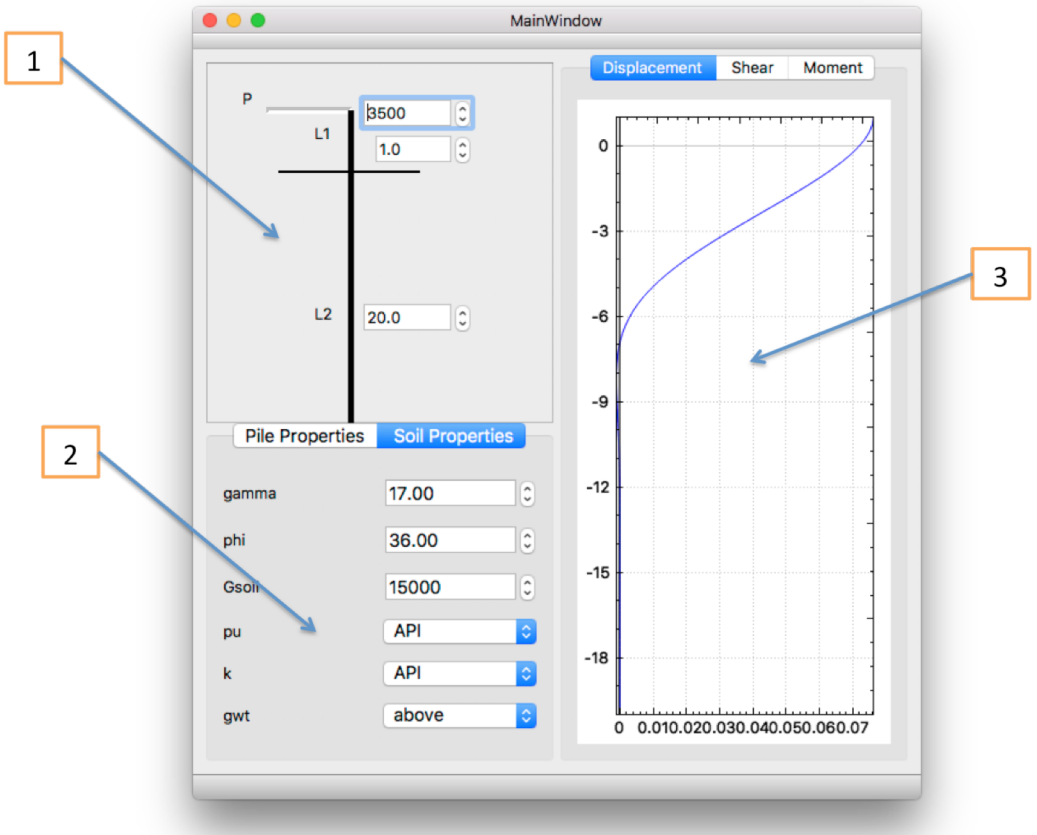
DESIGNING AN APPLICATION



- On the way to Version 0.1 of the PileGroupTool
  - First idea
  - Rough sketch of elements and layout

DESIGNING AN APPLICATION

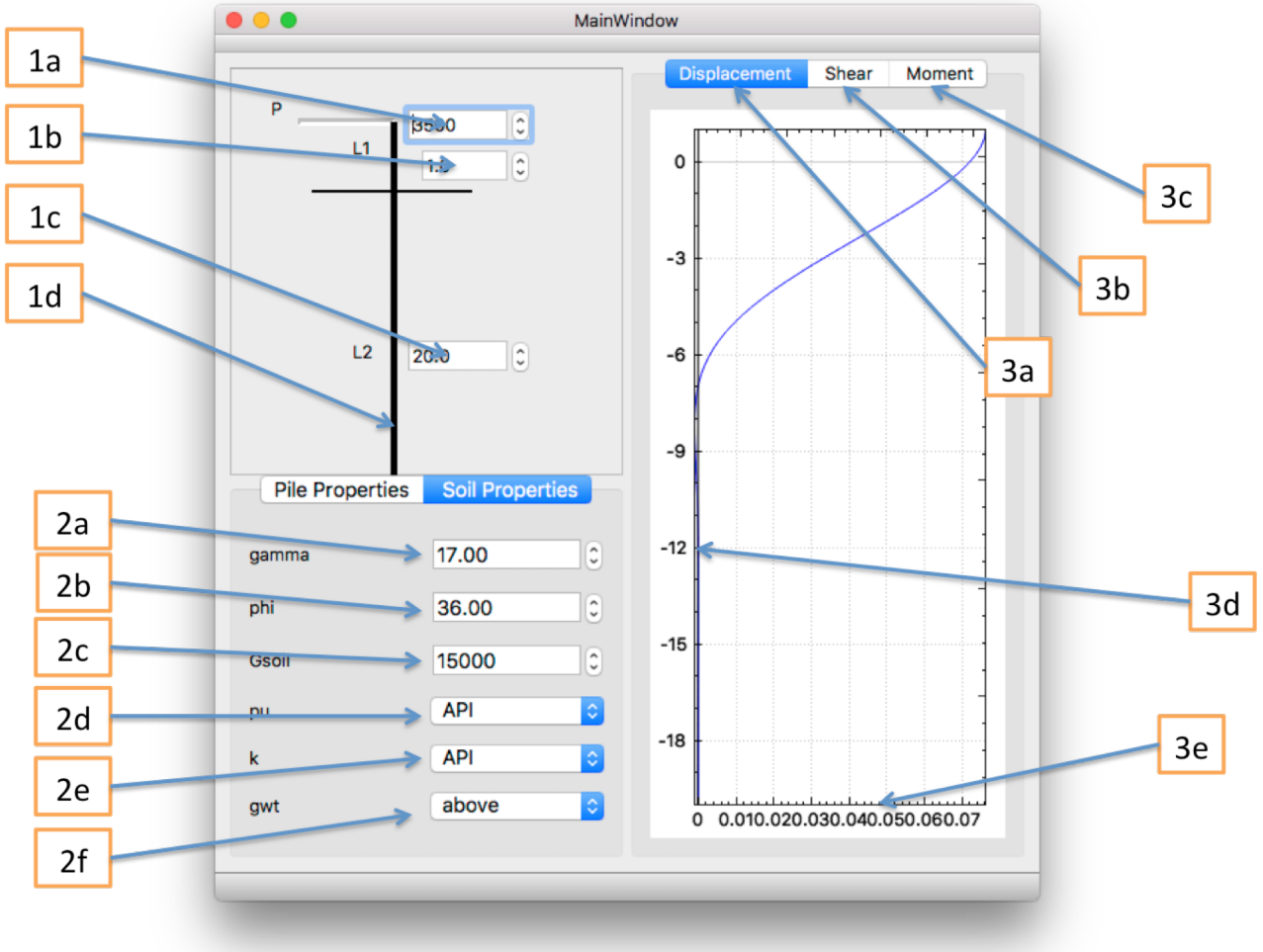
- Version 0.1 of PileGroupTool
  - Layout of groups / Widgets
  - Initial definitions



Element ID	Element	Description	Category	Action & Events	Else
1	Problem definition area		container		
2	Parameter definition area		notebook		
3	result visualization area		notebook		use instances of QCP

DESIGNING AN APPLICATION

- Version 0.1 of PileGroupTool
  - Identifying individual elements
  - Define type
  - Define functionality / actions if clicked/changed/ …



DESIGNING AN APPLICATION

Element ID	Element	Description	Category	Action & Events	Else
1	Problem definition area		container		
1a	applied horizontal force	textinput		store info and adjust plot in section 1	
1b	layer #1 thickness	textinput		store info and adjust plot in section 1	
1c	layer #2 thickness	textinput		store info and adjust plot in section 1	
1d	visualization/pile	graphic		double-click activates property seaction 2	
2	Parameter definition area		notebook		
2a	specific weight	textinput		update property variable upon change	
2b	friction angle	textinput		update property variable upon change	
2c	shear modulus	textinput		update property variable upon change	
2d	pu (ultimate pressure)	textinput		update property variable upon change	
2e	k-parameter	textinput		update property variable upon change	
2f	qround water table	combo box: above|below		update property variable upon change	defines whether we deal with saturated or wet soil
3	result visualization area		notebook		use instances of QCP
3a	displacement graph selector	visualize computed displacements	tab	change page in notebook to show respective result	
3b	moment graph selector		tab	change page in notebook to show respective result	
3c	shear graph selector		tab	change page in notebook to show respective result	
3d	pile position axis		QCP	allow to zoom in/out	measured from top down
3e	result value axis	adjust to max value	QCP		

EXERCISE #1: GUI DESIGN

- Design a UI for an application that collects a person's information
  - First and last name
  - Address, city, state, ZIP
  - Date of birth
- Create a table listing each element

ID	Type	Action	Widget	notes
1	Text input	none	???	Check for valid name?
2				

- Share with neighbor, discuss options, revise your design as appears useful

QT FRAMEWORK

What is Qt?

- A framework to
  - Create platform-independent applications
    - Desktop: Windows, Mac, Linux
    - Mobile devices: iOS, Android
    - Cars, Medical devices, …
  - Provide a large number of very useful data representation classes
- IT IS NOT FREE !!!!
  - Free for OpenSource
  - Free for personal use

COMMON DATA CLASSES

- QString

```
#include <Qstring.h>

QString mString;
```

  - A smart string object
  - No worries about '\0' (which is a pain even for experienced C-programmers, honestly)
  - Has formatting tools

```
mString = "this is process {} of {}";

mString.arg(proc).arg(numProcs);
```

  - Has Unicode support (Asian fonts, European fonts)

COMMON DATA CLASSES

- QVector<TYPE>
  - QVector<double> array1;
  - QVector<double> *array2 = new QVector<double>();
  - QVector<QVector<double> *> array3;

- array1.append(42.0);
- int n = array2->size();
- double x = array1[2]; array1[1] = array1[2]; array1[1] = x;
- array3[2] = new QVector<double>();

COMMON DATA CLASSES

- QList<TYPE>
  - QList<QString> stringList1;
  - QStringList stringList2;

- Looping made simple:

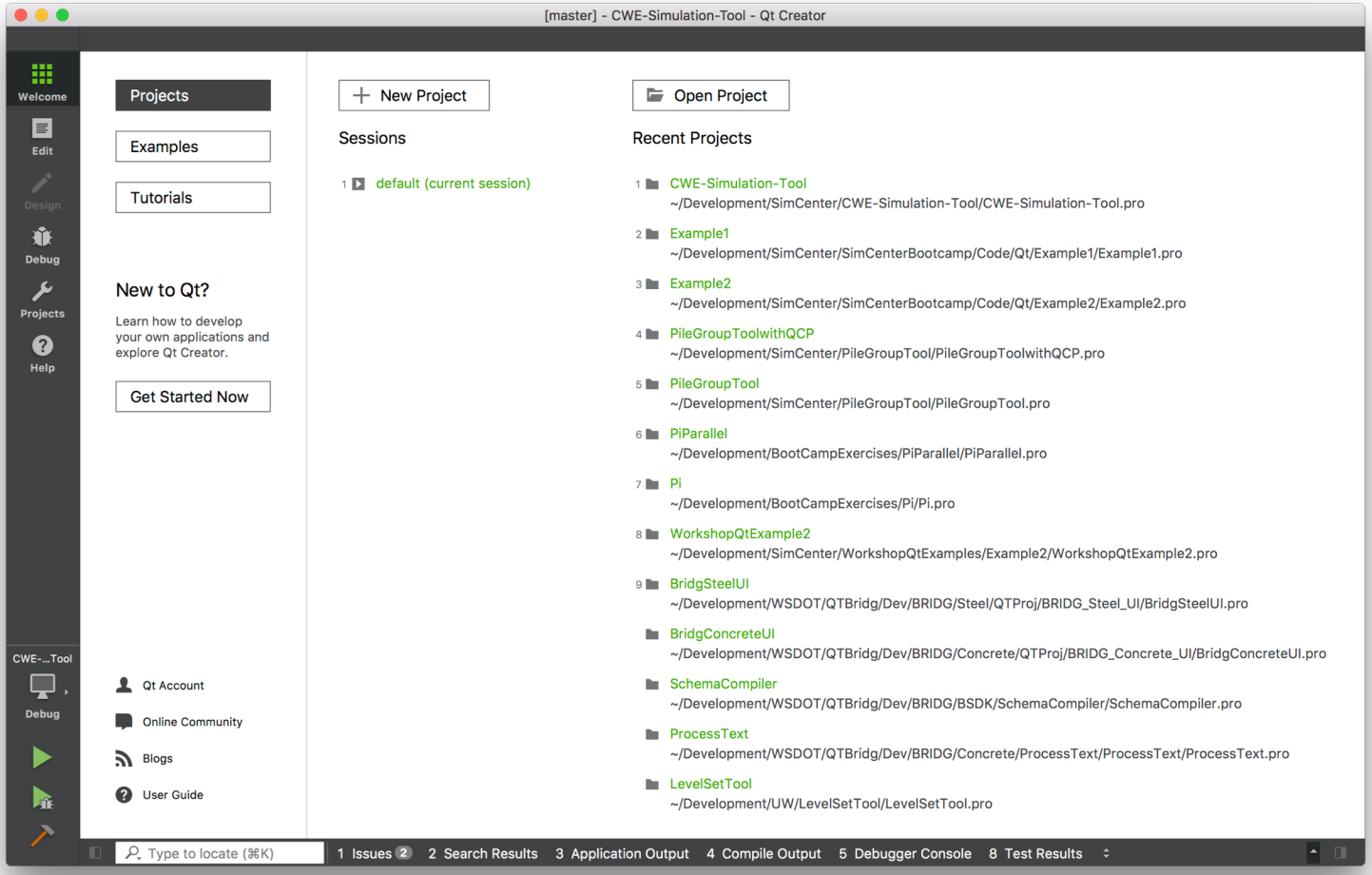
```
#include <iostream.h>
#include <Qstring.h>
#include <QStringList.h>

foreach (QString s, stringList1) {
        // do something with string s
        std::cout << s << std::endl;
}
```

BUILDING THE GUI

- **Option #1:**
  - Directly in code
  - Check out http://zetcode.com/gui/qt5/ **(THESE GUYS ROCK !)**
- **Option #2:**
  - Using Qt Designer (built into Qt Creator)
  - Let's switch and build your app together **(Live Demo)**

DEVELOPER TOOL FOR QT

- Qt Creator



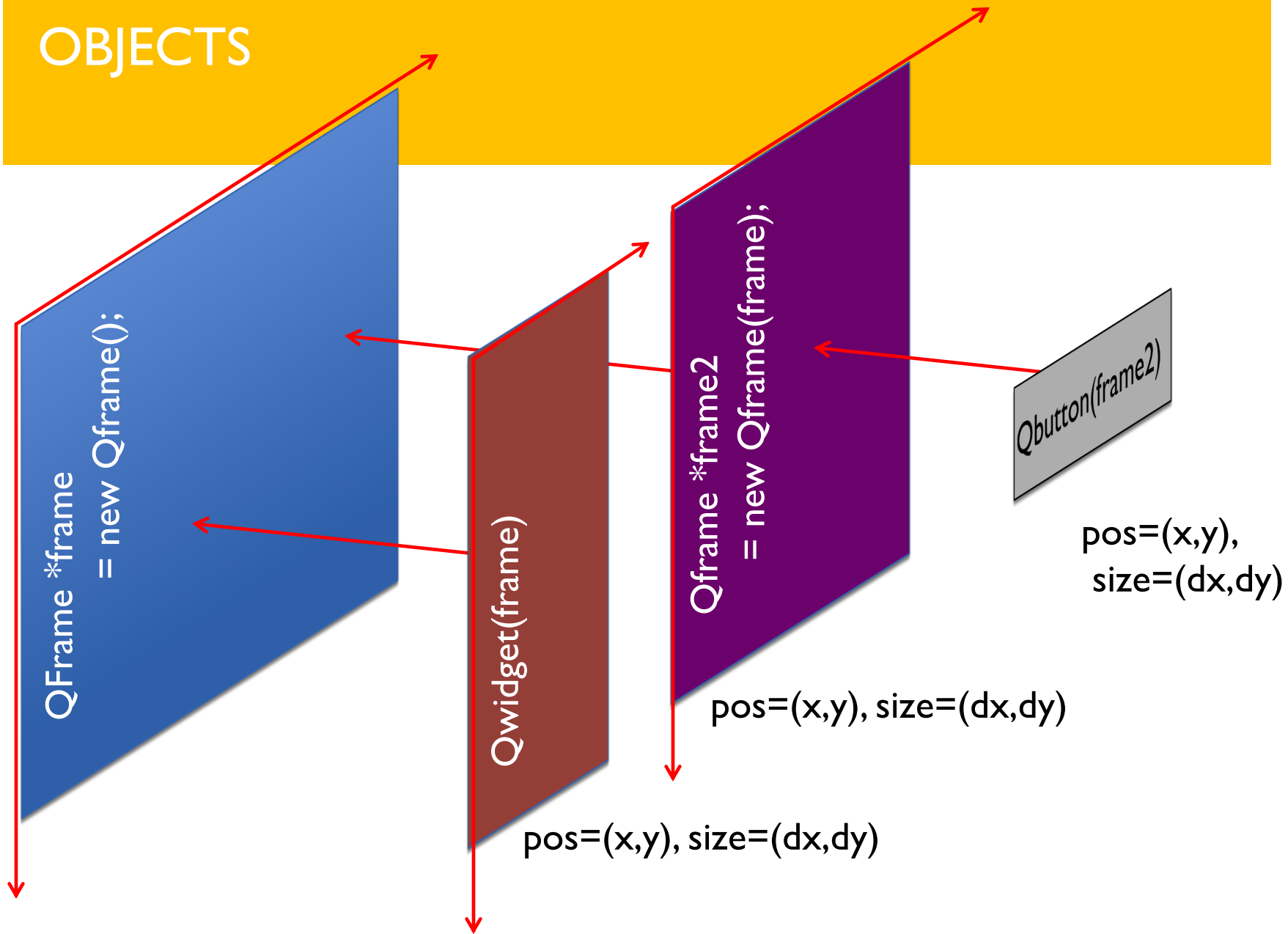
EXERCISE #2: CREATING YOUR GUI

- Let's return to your GUI design from Exercise #1

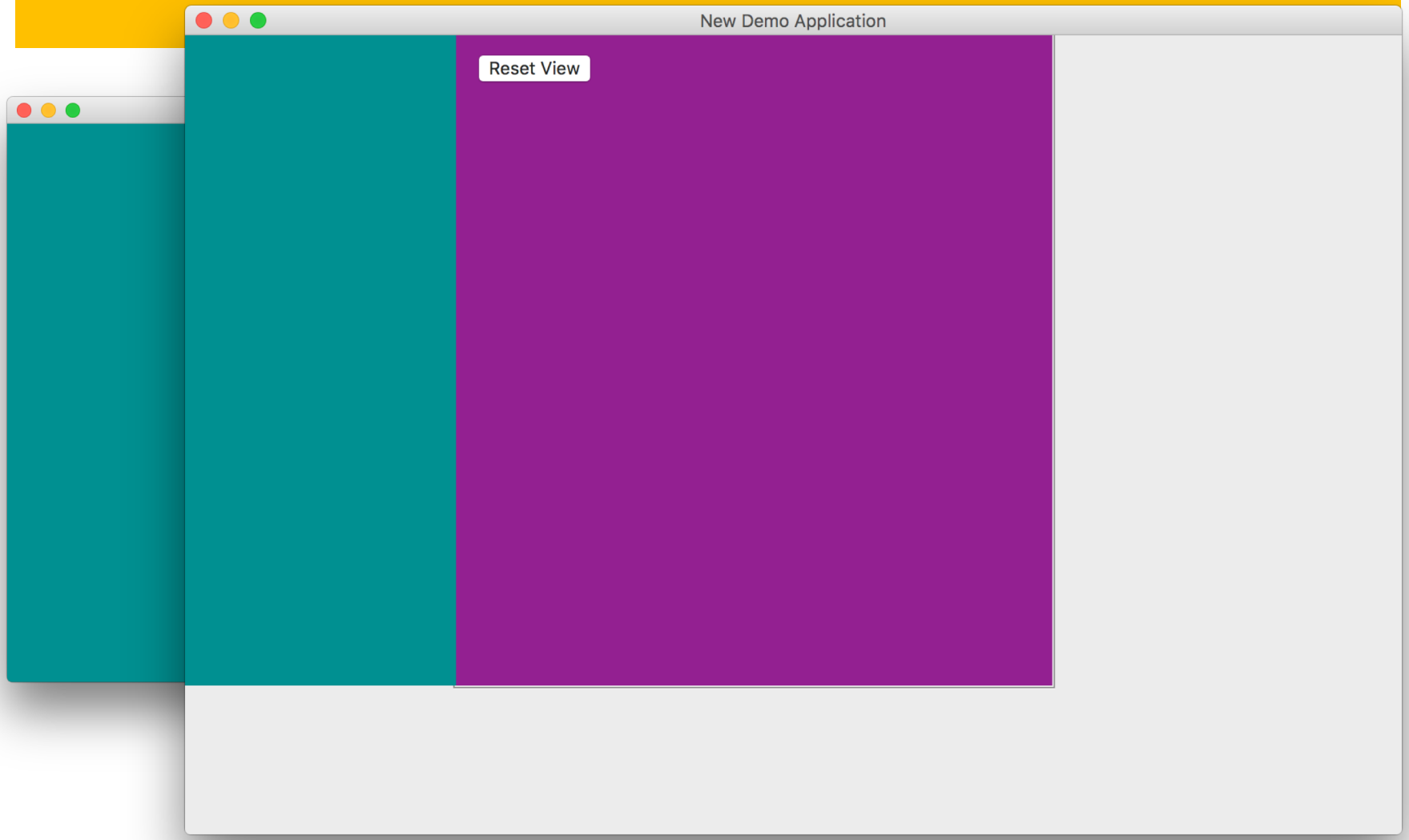
1. Create a new Qt Widget Application project using Qt Creator
2. Open Forms => MainWindow.ui
3. Create your GUI as close to your design as possible
4. Go through all the objects and assign them a more descriptive name like:
   - TB_firstName
   - CB_theState
   - Etc.
5. Run qmake, build the app, and run it

This one should be surprisingly easy ☺

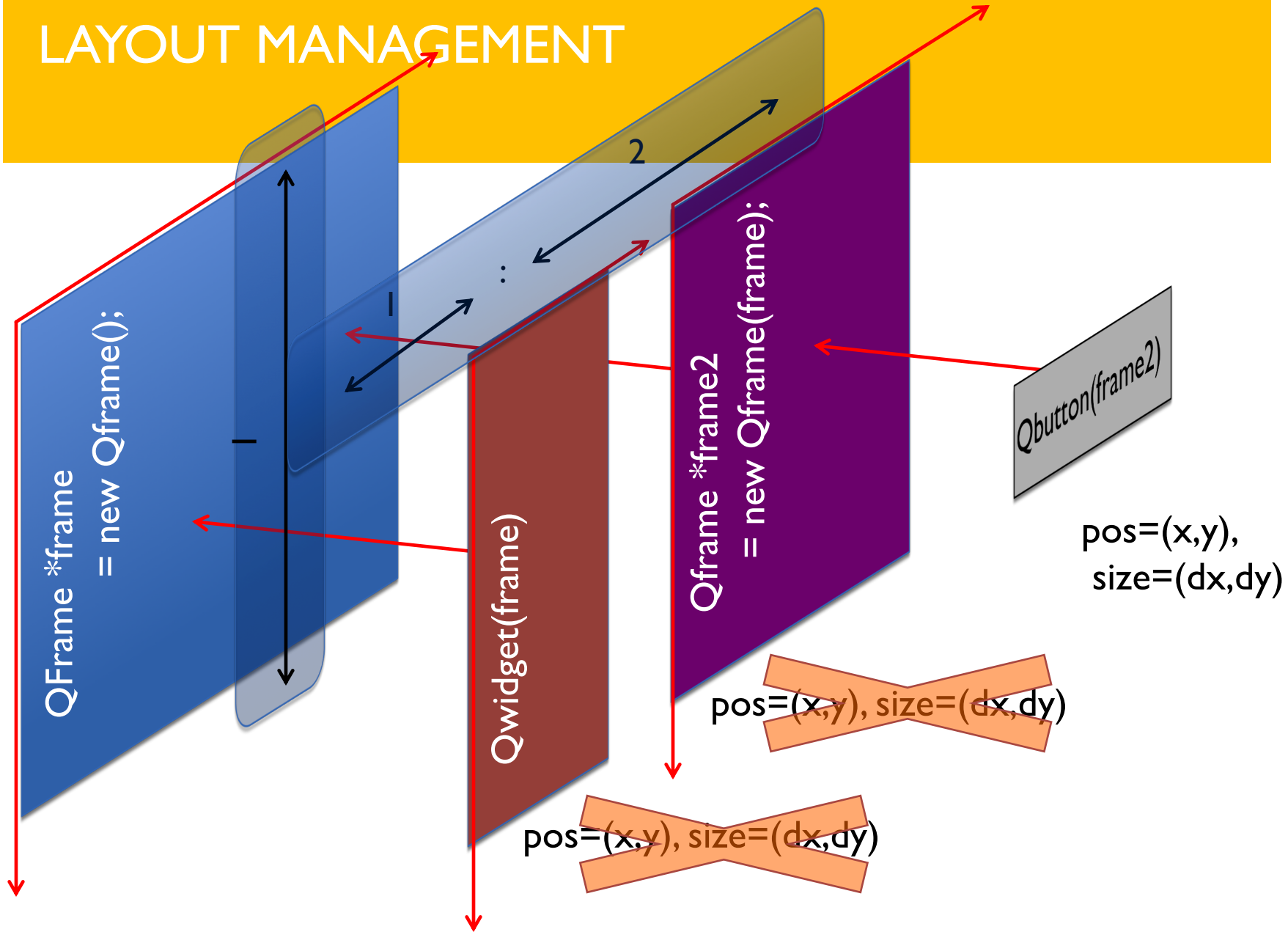
OBJECTS


QFrame *frame
= new Qframe();
Qwidget(frame)
Qframe *frame2
= new Qframe(frame);
Qbutton(frame2)
pos=(x,y),
size=(dx,dy)
pos=(x,y), size=(dx,dy)
pos=(x,y), size=(dx,dy)

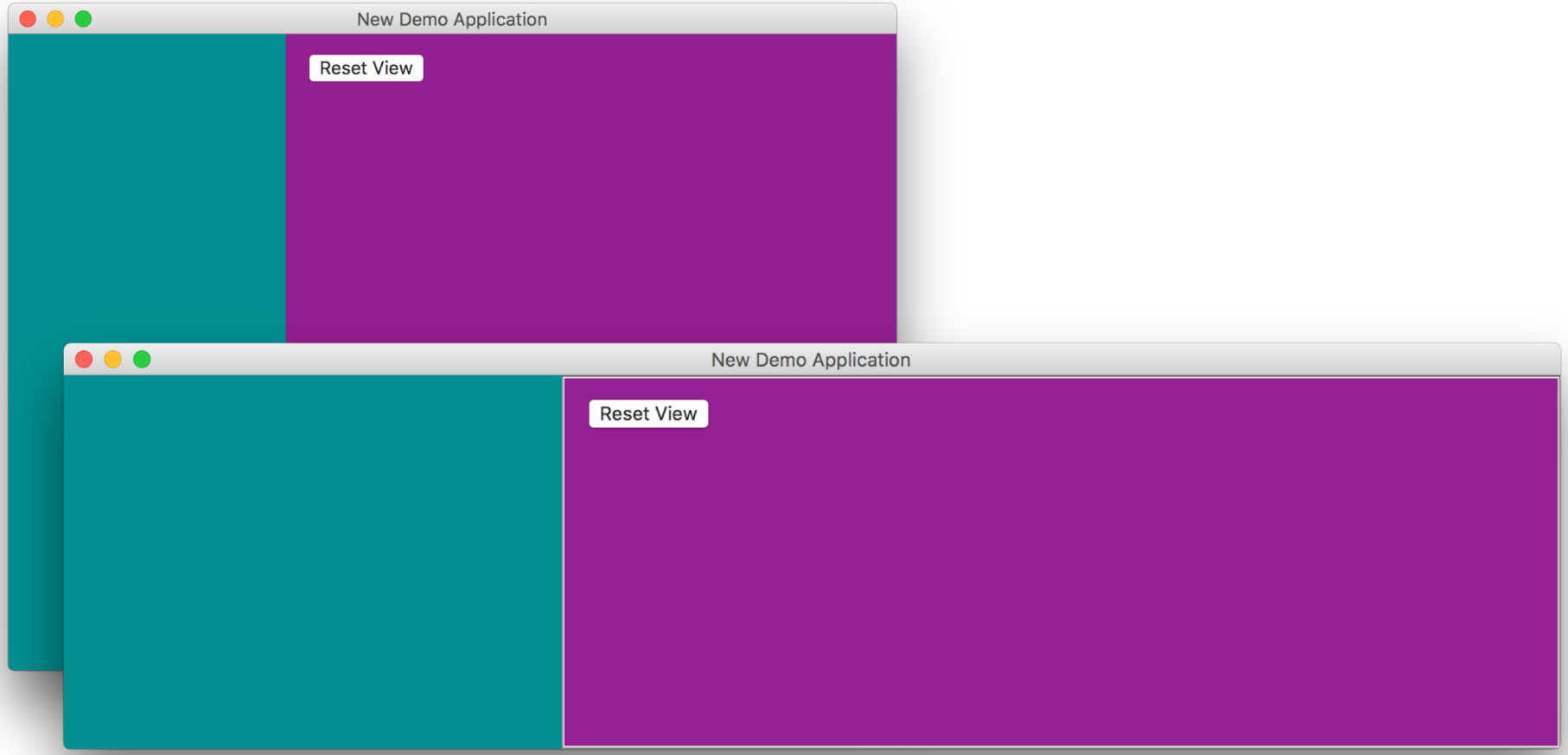
A SIMPLE APPLICATION



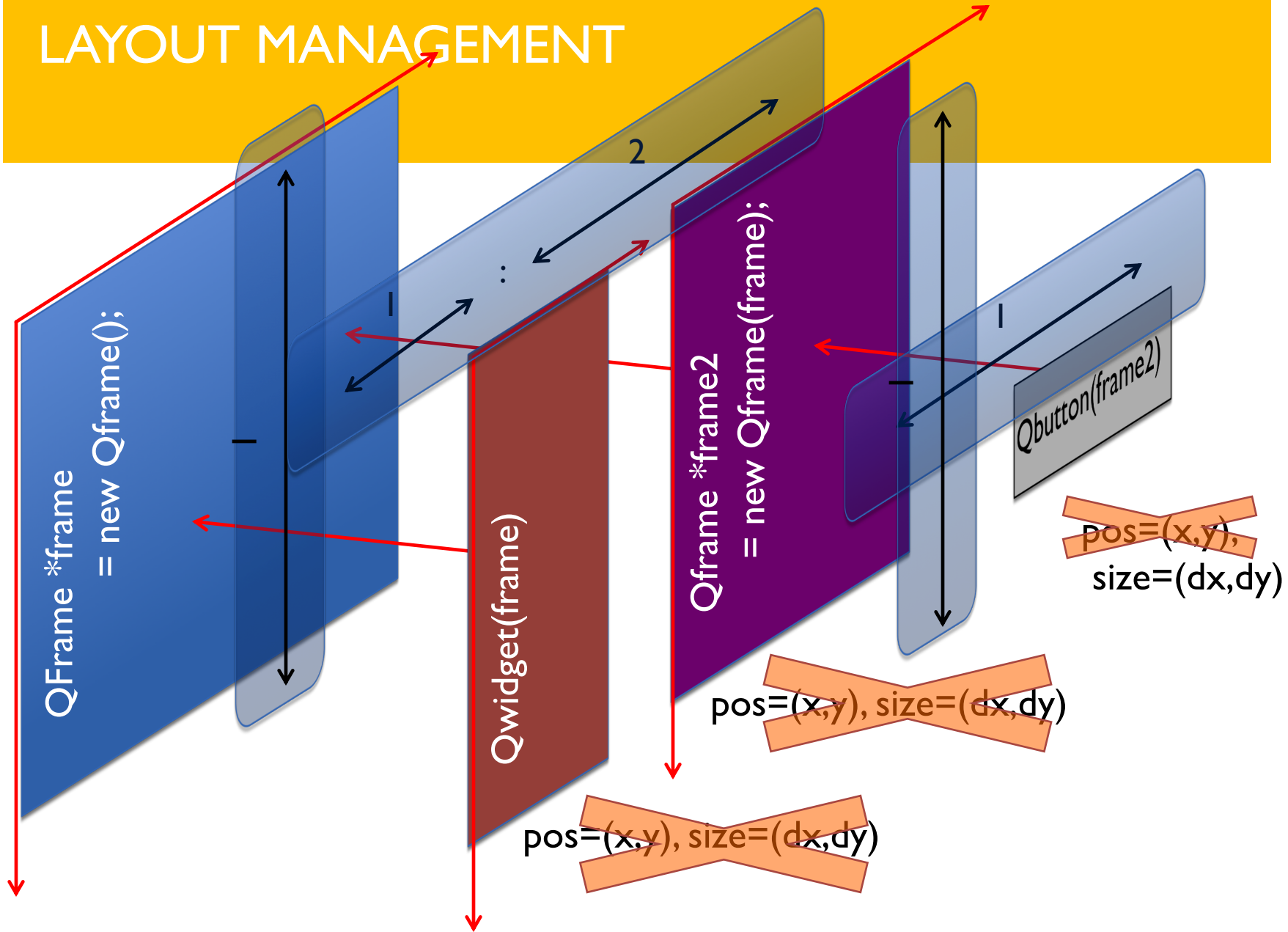
LAYOUT MANAGEMENT



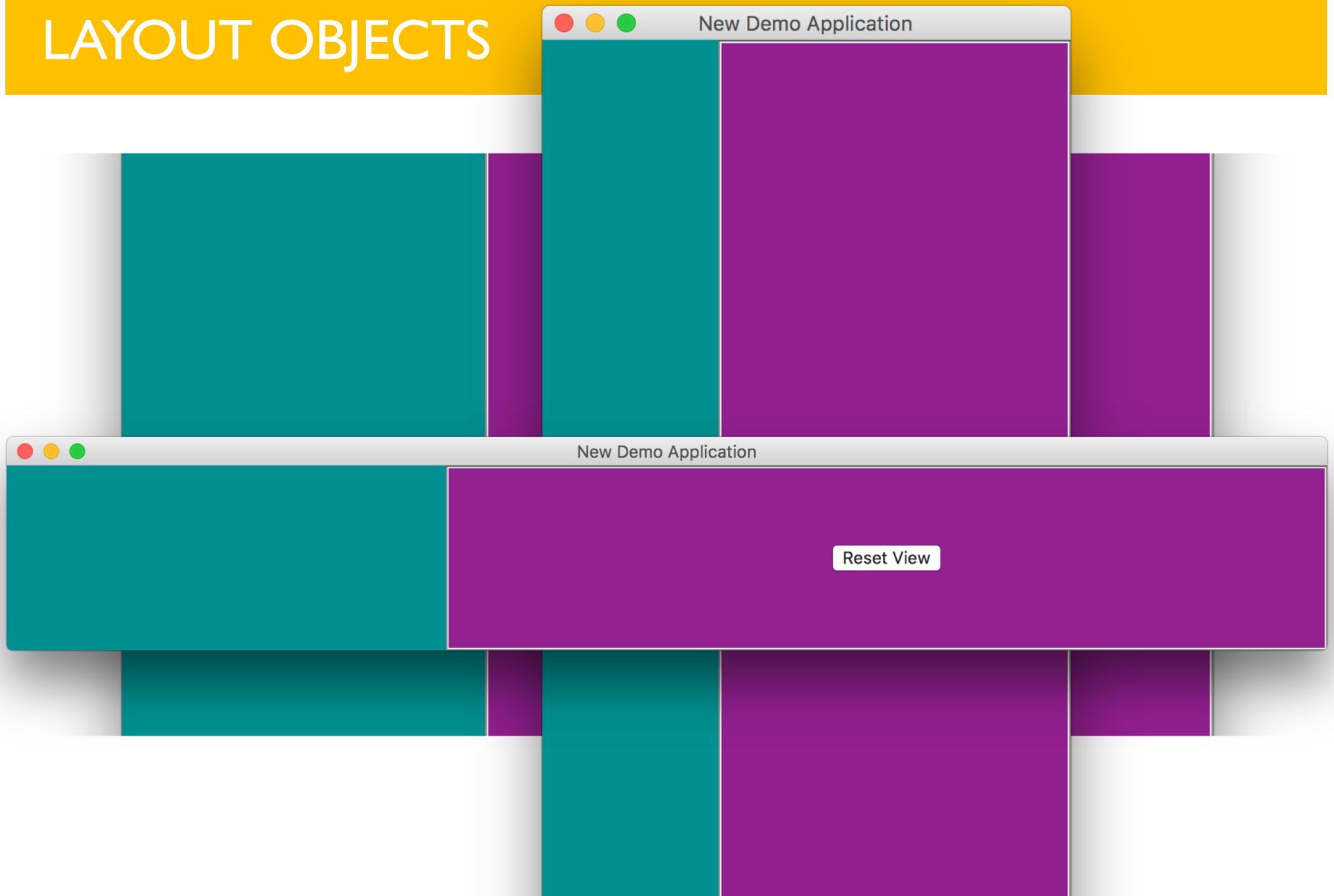
A SIMPLE APPLICATION USING LAYOUTS



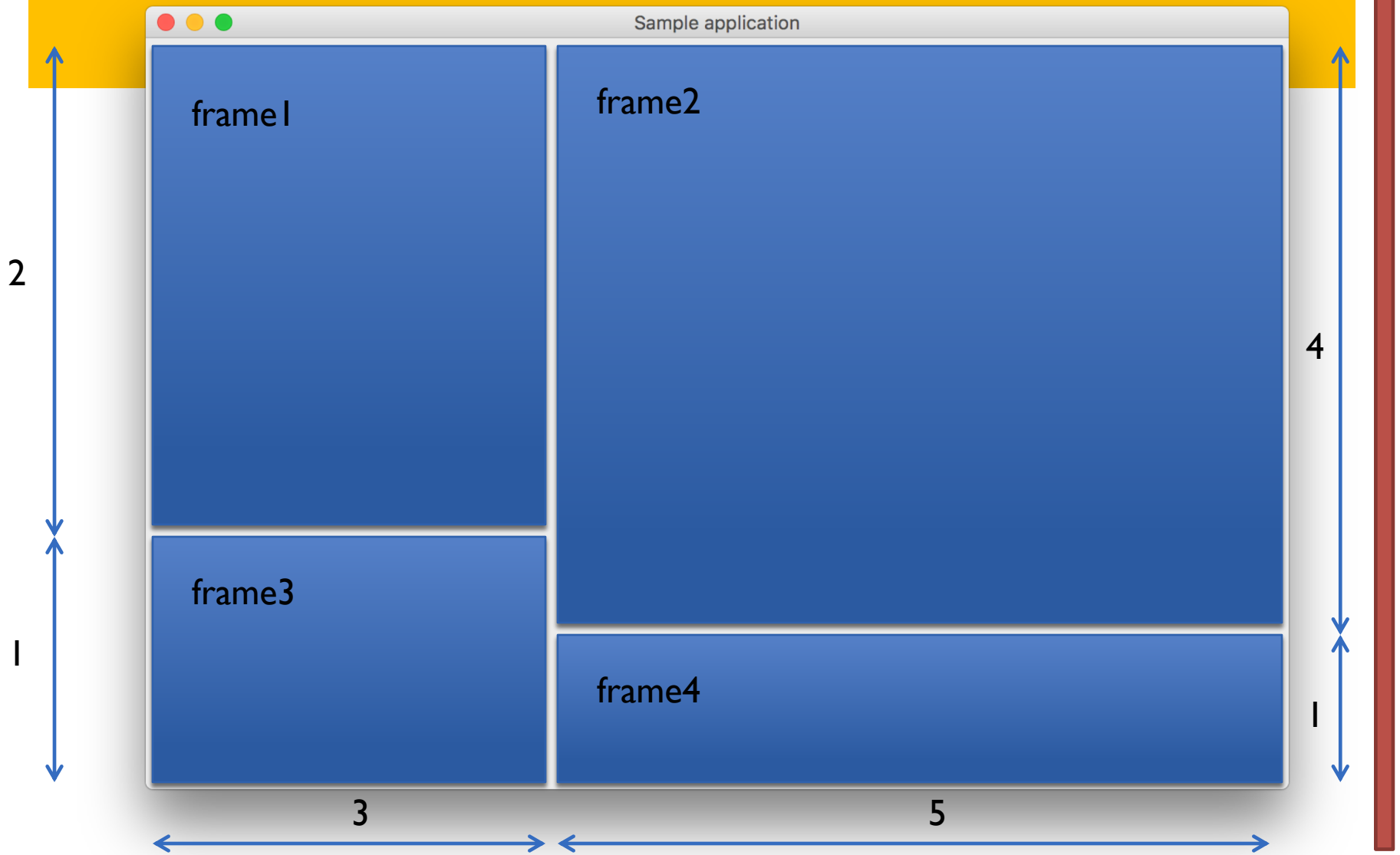
LAYOUT MANAGEMENT



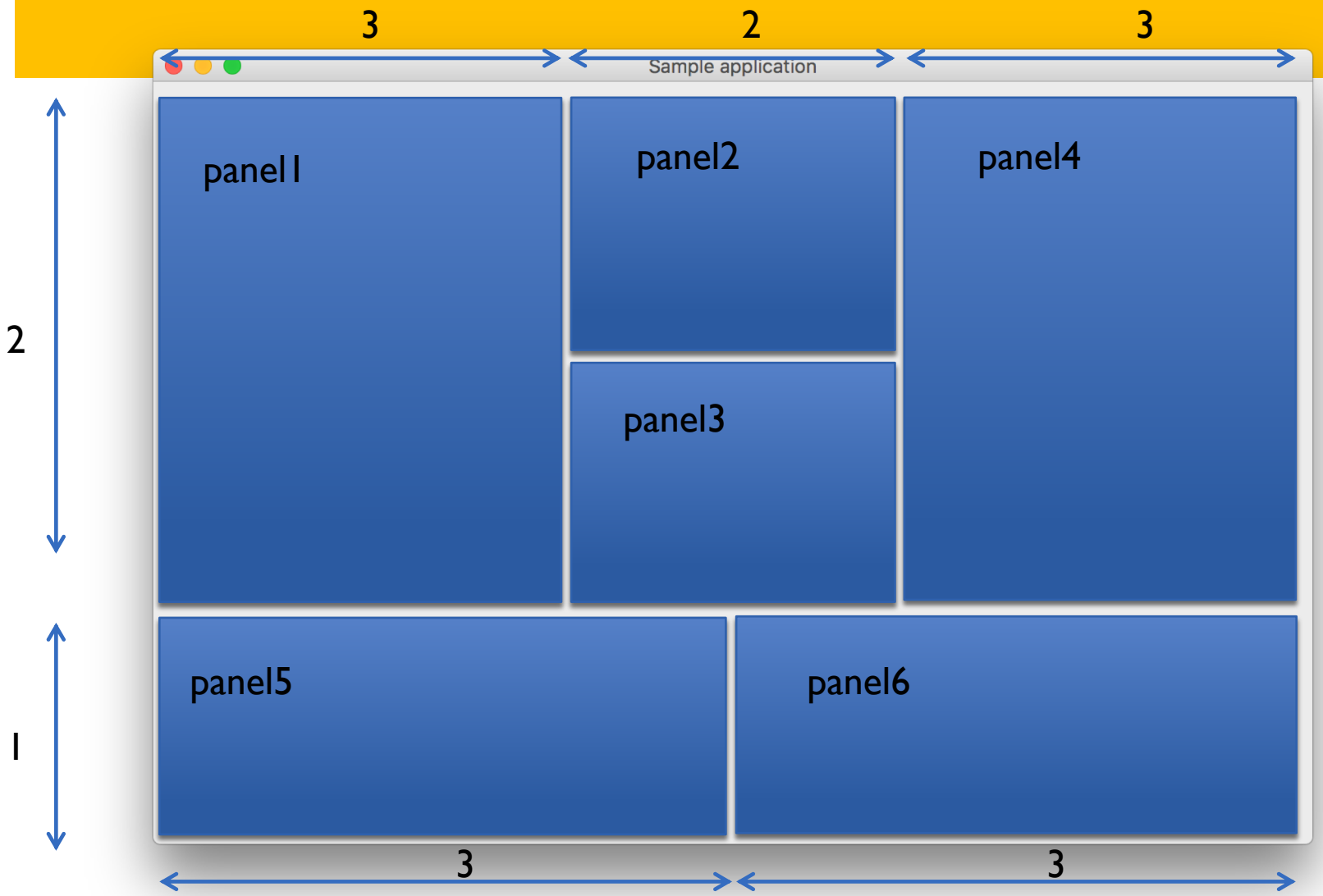
A SIMPLE APPLICATION USING MULTIPLE LAYOUT OBJECTS



EXERCISE #3A: LAYOUTS



EXERCISE #3B: IN CASE THE ORIGINAL PROBLEM WAS TOO SIMPLE



EXERCISE #4: CREATING YOUR GUI

- Let's return to your GUI design from Exercise #2

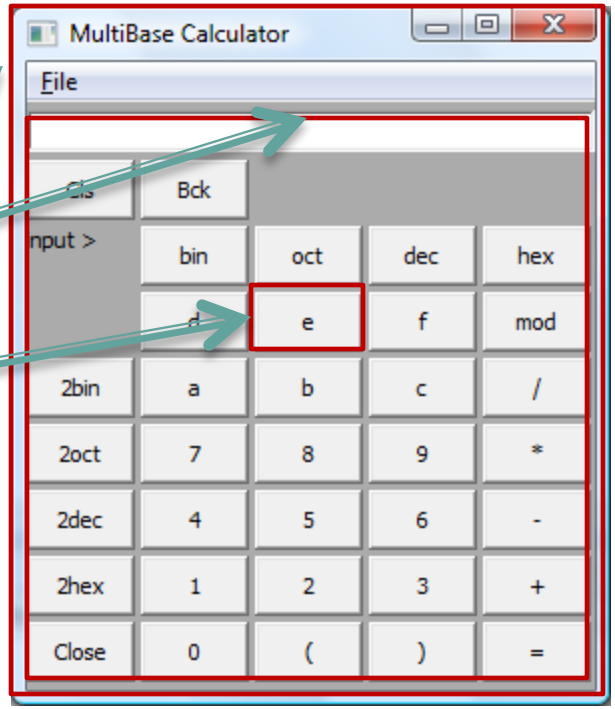
1. BEFORE doing anything, think about layout for your app.
   - How do you want each field to line up?
   - How shall each field grow relative to each other?
   - How can you achieve that with the least of layouts?
2. Move on and implement your layout
   1. Select container object
   2. Right-click and select layout
   3. Choose the desired layout

This one is usually harder but VERY IMPORTANT

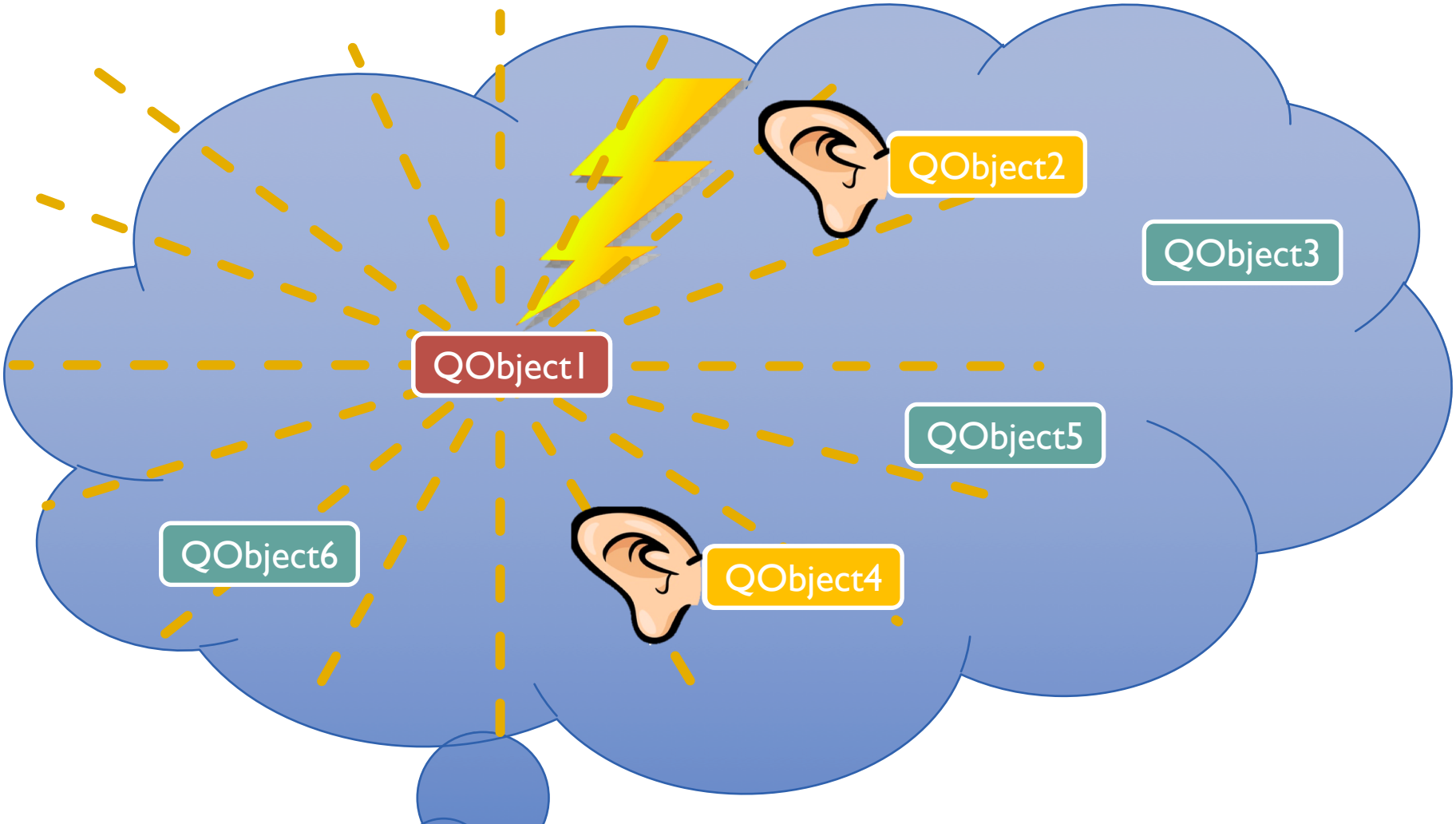
SIGNALS AND SLOTS

How does a GUI work?

- Create the graphics
  - Instance of QMainWindow
  - Add child widgets
    - QFrame
    - QPushButton
    - etc.
- Emit signals for events
- Connect signals to slots
- Run the Event loop



SIGNALS AND SLOTS



OPTION 1: OVERLOADING DEFAULT SLOTS

- Each Widget emits signals on specific events

Signals

void	**clicked**(bool *checked* = false)
void	**pressed**()
void	**released**()
void	**toggled**(bool *checked*)

- 3 signals inherited from QWidget
- 2 signals inherited from QObject

- Each widget has a unique name
- Example:
  - Widget name: run_button
- Event clicked connects to default slot:
  - on_run_button_clicked()
  - You can overload that slot in your application
- Implementation made easy:
  - Qt Creator
    - Right click => go to slot => clicked

OPTION 2: CREATING YOUR OWN SLOTS

```
// app.h: definition
…
class MyClass {
…
private slots:
   void react_to_button_clicked();
}
```

```
// app.cpp: implementation

void MyApp::react_to_button_clicked() {
            // your response to button clicked
}
```

```
// app.cpp: constructor

void MyApp::MyApp(Qobject *parent = 0) {
    ….
    // connect signal to slot
    connect(ui->myButton, SIGNAL(on_myButton_clicked()),
                    this, SLOT(react_to_button_clicked()));
}
```

pointer to source

Signal function

pointer to receipient

slot/callback function

OPTION 2: CREATING YOUR OWN SLOTS

```
42
43  private slots:
44      //
45      // program controls
46      //
47
48      // menu actions
49      void on_actionExit_triggered();
50      void on_actionNew_triggered();
51      void on_actionSave_triggered();
52      void on_action_Open_triggered();
53      void on_actionExport_to_OpenSees_triggered();
54      void on_actionReset_triggered();
55
```

```
462 /* ***** menu actions ***** */
463
464 void MainWindow::on_actionExit_triggered()
465 {
466     this->close();
467 }
468
469 /* ***** check box status changes ***** */
470
471 void MainWindow::on_chkBox_assume_rigid_cap_clicked(bool checked)
472 {
473     assumeRigidPileHeadConnection = checked;
```

```
78  void CWE_file_manager::linkMainWindow(CWE_MainWindow *theMainWin)
79  {
80      CWE_Super::linkMainWindow(theMainWin);
81      if (!cwe_globals::get_CWE_Driver()->inOfflineMode())
82      {
83          ui->remoteTreeView->setModelLink(theMainWindow->getFileModel());
84          QObject::connect(ui->remoteTreeView, SIGNAL(customContextMenuRequested(QPoint)),
85                           this, SLOT(customFileMenu(QPoint)));
86          QObject::connect(cwe_globals::get_file_handle(), SIGNAL(fileOpDone(RequestState,QString)),
87                           this, SLOT(remoteOpDone(RequestState,QString)));
88          QObject::connect(cwe_globals::get_file_handle(), SIGNAL(fileOpStarted()),
89                           this, SLOT(remoteOpStarted()));
90          setControlsEnabled(true);
91      }
92  }
```

```
603 }
```

DEBUGGING WITH SIGNALS AND SLOTS

- Clean way

1. Set breakpoints at entries to slot implementation(s)
2. Start ("run") application
3. Don't stop at first occurrence but continue till app accepts new user input.

- Brute-force method:
  - Write debug output at start of slot implementation(s)

```
#include <QDebug.h>

void MyClass::MySlot(int arg1) {
        qDebug() << "Entering MySlot";
        // your code here
}
```

EXERCISE #5: ADDING CALLBACK FUNCTIONS

- Let's add some functionality to your GUI

1. Create a class method (function) that collects the information from the UI and stores it in a private structure like this one:

```
typedef struct {
        QString firstName;
        QString lastName;
        …
} DATA;
```

2. Create a slot that writes out a formatted address label
3. Create a button (if you don't have one yet) labeled "Print Address Label"
4. Connect this button's clicked signal to your slot
5. Qmake => build => run

DESIGN CONSIDERATIONS

- VIEW – CONTROLLER – DATA model
  - VIEW
    - Visual parts, display classes
  - CONTROLLER
    - Registers user requests
    - Manages actions in analysis models
    - Controls flow of data
  - DATA
    - All kinds: text, floats, arrays, class objects, …

This is the image of your app

This represents the smarts of your app

This is what only Excel users care to look at

MODEL – VIEW CONCEPT

- QAbstractItemView
  - QTreeView
  - QTableView
  - QListView

The Display Widget

```
QTreeView mView;
```

- QAbstractItemModel
  - QAbstractItem

The data to be displayed

```
QAbstractItemModel *model = new QAbstractItemModel();
```

- Connecting data and view:

```
mView.setModel(model);
```

Note: this is just a pointer to the model, NOT a copy.

USEFUL HELPER WIDGETS

- QDialog
  - QFileDialog
  - QMessageDialog
  - QColorDialog
  - QFontDialog
  - …
- QDir … all the help you need dealing with paths across different platforms
- QDateTime … dealing with time formats, date formats, calculating number of days, elapsed time, time zones

EXERCISE #6: CREATE A NICE ADDRESS LABEL

1. Update your slot for create_label_button_clicked() (or add another one) such that is
   - Pops open a dialog showing a nicely formatted address label in a QTextBrowser widget