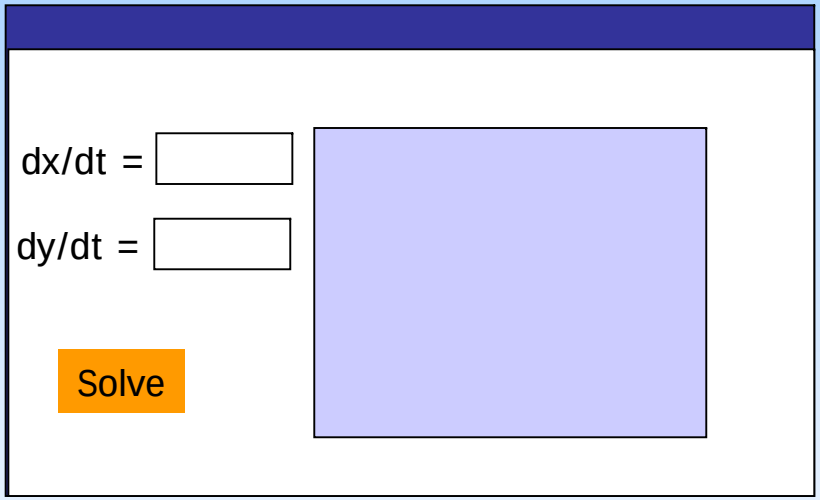
Math 157

Creating Qt GUI Applications.

Feb. 16, 2009

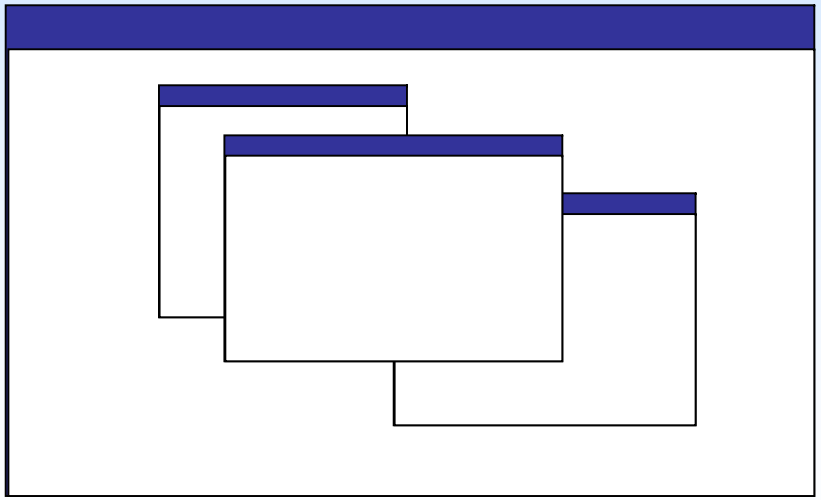
Qt Applications

Single Document Interface (SDI)



We’re creating SDI applications

Multiple Document Interface (MDI)



Qt SDI Application …

How do we think we could create one?

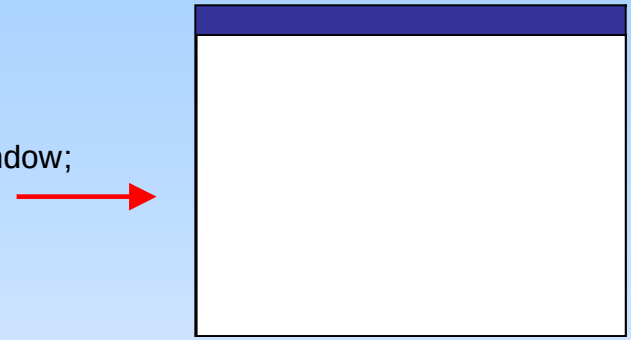
```
main()
{

   general setup
   create the application window

   loop while the window exists
    {
       check window for events
       process events
    }


}
```

Qt SDI Application …

How is it actually done?

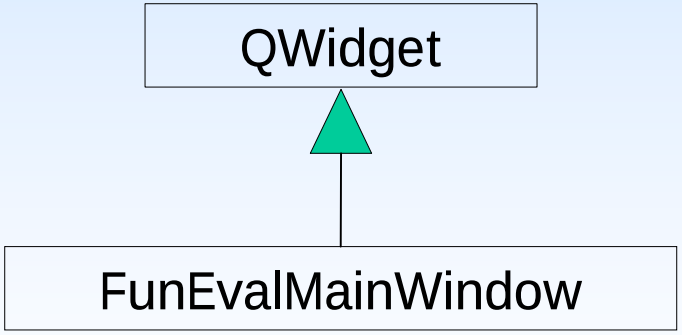
```
main()
{
    general setup
    create the application window
    loop while the window exists
    {
    check window for events
    process events
   }
}
```

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    FunEvalMainWindow mainWindow;
    mainWindow.show();


    return app.exec();


}
```



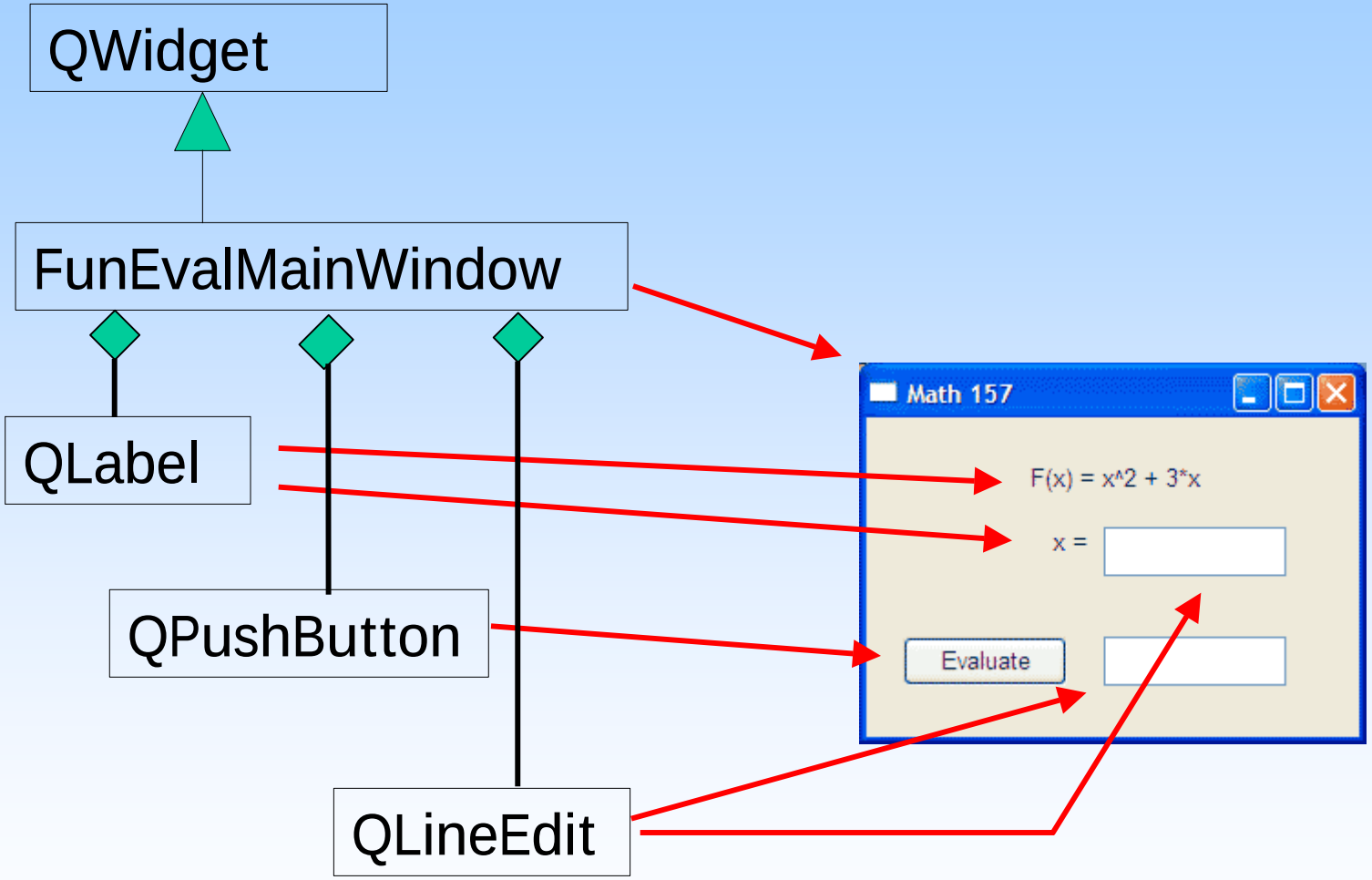
QApplication

Contains event loop



Contains code for main window

Function Evaluator Main Window



Function Evaluator Main Window

- Class structure “reflects” window appearance
- Implement aggregate associations using pointers

(1) Main window class declaration:

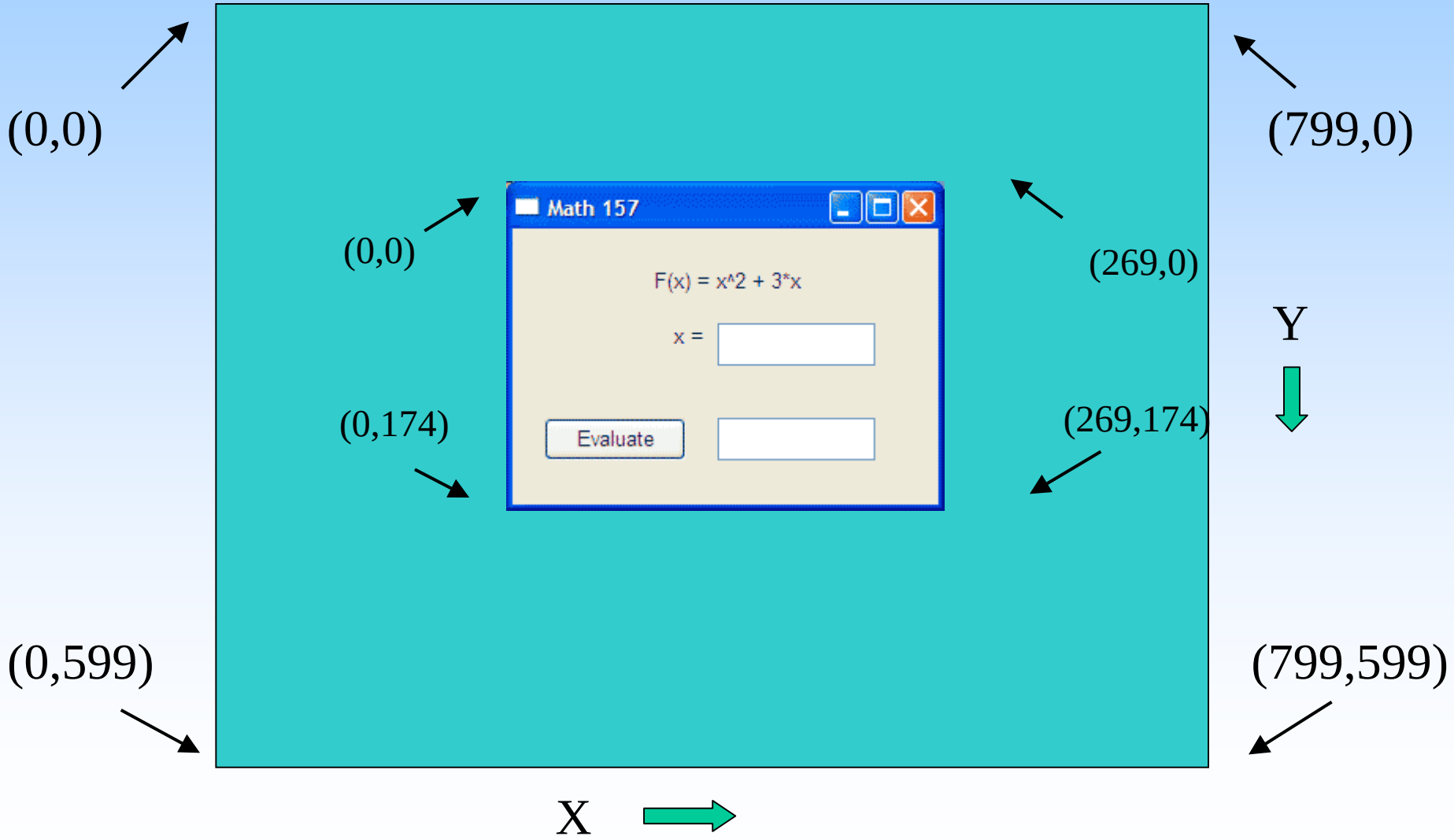
Declare pointers to child widgets as data members

(2) Main window constructor:

Use **new** to instantiate child widgets. Use main window’s this pointer as parent argument

Screen/Windows Coordinate System

Screen in 800X600 mode



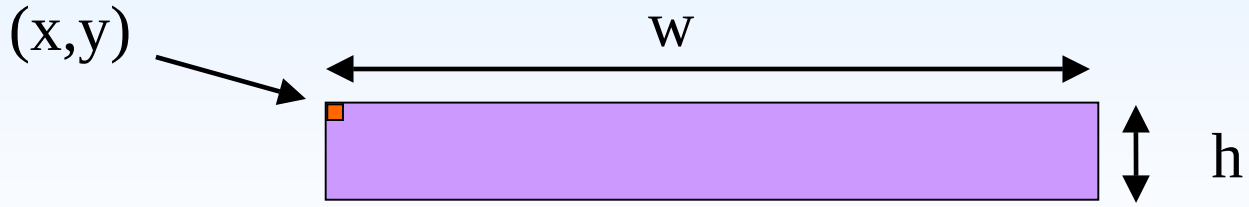
Screen/Windows Coordinate System continued ...

The main window is placed with respect to the *screen* coordinate system.

Child widgets are placed with respect to the *window* coordinate system.

One can use QRect objects to specify rectangular regions.

QRect(int x, int y, int w, int h)



Application Event Handling

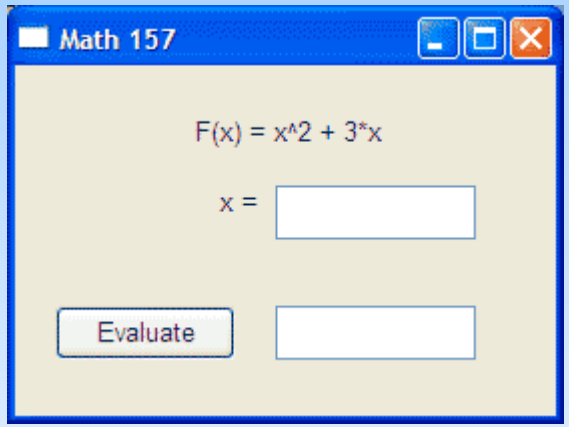
Application Class

{

Event Loop

}

Window Class



Evaluate button hit

OnEvaluate() called

```
OnEvaluate()
{
    *****
    *****
}
```

Function Evaluator Event Setup

Ingredients:

QWidgets emit “signals” - signals specified in documentation

QWidgets define “slots” - response functions

QWidgets create connections between signals and slots

Procedure:

Create a response function (slot) in the FunEvalMainWindow class

Identify the signal emitted by the child widget

Connect the signal to the slot

Function Evaluator Application Event Setup

In FunEvalMainWindow.h declare OnEvaluate() response functions as a slot:

```
class FunEvalMainWindow : public QWidget
{
            *
            *
    QPushButton* evaluateButton;
            *
            *
    public slots:

    void OnEvaluate();
};
```

In FunEvalMainWindow.cpp define the OnEvaluate() response function.

```
void FunEvalMainWindow::OnEvaluate()
{
//  read x value (as a Qstring)
//
    QString readText = xTextEdit->text();
                *
                *
}
```

Application Event Setup

In the FunEvalMainWindow constructor create the connection:

```
FunEvalMainWindow::FunEvalMainWindow()
{
            *
            *
//
// Connect evaluateButton clicked() signal to "this" OnEvaluate()
//
QObject::connect(evaluateButton,SIGNAL(clicked()), this, SLOT(OnEvaluate()));
```

Discover the signal emitted by reading the documentation for QPushButton:

QPushButton

```
          *
          *
```

Additional Inherited Members

4 signals inherited from QAbstractButton

QAbstractButton

```
          *
          *
```

Signals

```
void clicked ( bool checked = false )
void pressed ()
void released ()
```

Source code procedure for “hand-crafted” Qt application

(1) Create main(...) routine containing the QApplication instance and main window instance.

(2) Create main window class derived from QWidget

In the class declaration

- declare **Q_OBJECT** macro
- declare pointers to child widgets as data members
- declare slots (response functions)

In the class source (constructor)

- layout main window
- instantiate child windows using new
- create connections from signals to slots

In the class source

- implement response functions

Function Evaluator File Structure

FunEvalApp.cpp

main(...)
QApplication instance
Main window instance
app.exec()

FunEvalMainWindow.h

FunEvalMainWindow declaration
that also includes

Q_OBJECT	(A required Qt macro)
public slots:	(Qt specific construct)

FunEvalMainWindow.cpp

FunEvalMainWindow source code

What you need to do for Assignment 7

- Download and get the function evaluator application (original) running.
- Replace the “F(x) = X^2 + 3*x” by “F(x)= ” .
- Add a QLineEdit text edit box to capture the function string input.
- Change the code in OnEvaluate() to read the input function string, evaluate the function specified by the string, and then output the answer to the QLineEdit text box, resultTextEdit.
- Set up a response function and appropriate connections to to clear the results box when text is entered into the F(x) or x edit boxes.

Creating and Managing a Qt application

Initial setup

- qmake -project -o *TargetExectuable*
- Edit *.pro file, change TEMPLATE from **app** to **vcapp** and add the *.pro file to the SOURCES list (Visual Studio only)
- run qmake
- add the project created to your Math 157 Visual Studio solution

Usage

- build the project following the standard procedure
- modifications to the project (addition of include paths, source files, libraries, etc.) must be made to the *.pro file and followed by re-running qmake and accepting the reloaded project.
- You can add qmake as a “Tool” to expedite re-running qmake