ECE3574 - Model/View Programming

antoniob@vt.edu

(Graphical) User Interfaces

- User Interfaces
  - **Represent data**
  - Provide a medium to interact with the user
- Graphical User Interfaces are built with multiple graphical objects
  - Before design patterns
    - Graphical objects and the data they represent were lumped together
    - When the developer updates the graphical object have to rewrite the entire data handling
    - When the developer wants to reuse the same graphical object for some other data have to rewrite the entire graphical object

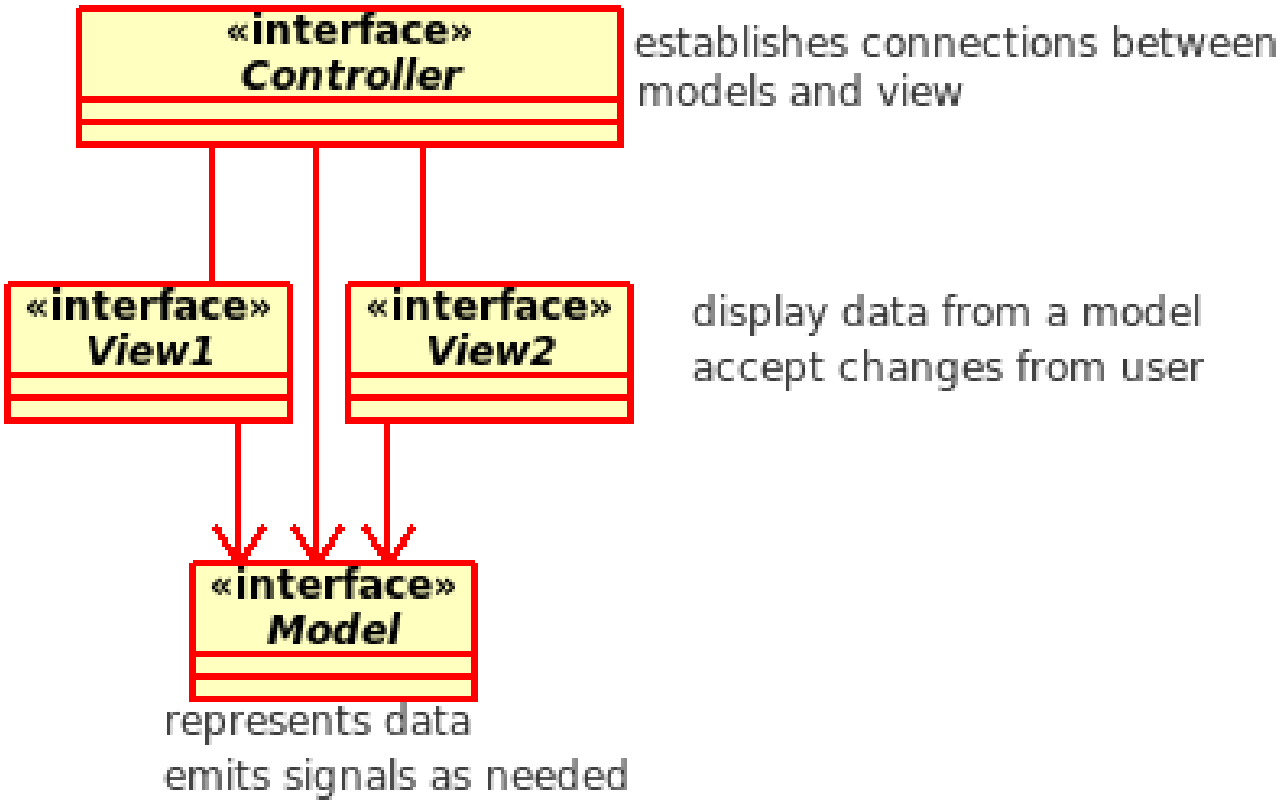
(Graphical) User Interface

- Design pattern
  - Good code practice
- Separate
  - The Data
  - From its presentation
- Model-View-Controller
  - Model/View Framework (Qt)

Model-View-Controller (MVC) 1/3

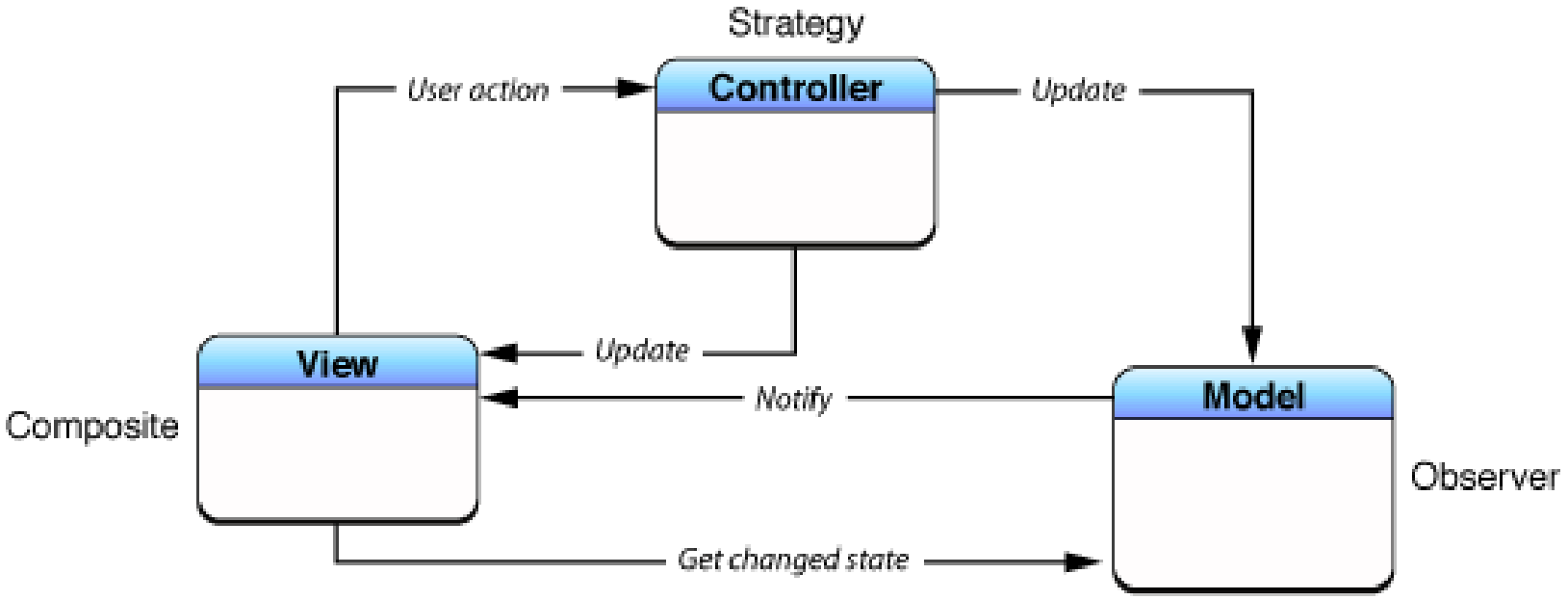
- **MVC** Consists of three kinds of objects
  - Model
    - Application object
  - View
    - Screen presentation
  - Controller
    - Defines the way the user interface reacts to user input
- Defined by [Gamma95]
- Originating from Smalltalk language

Model-View-Controller (MVC) 2/3



Model-View-Controller (MVC) 3/3

- Traditional version of MVC as a compound pattern



https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html

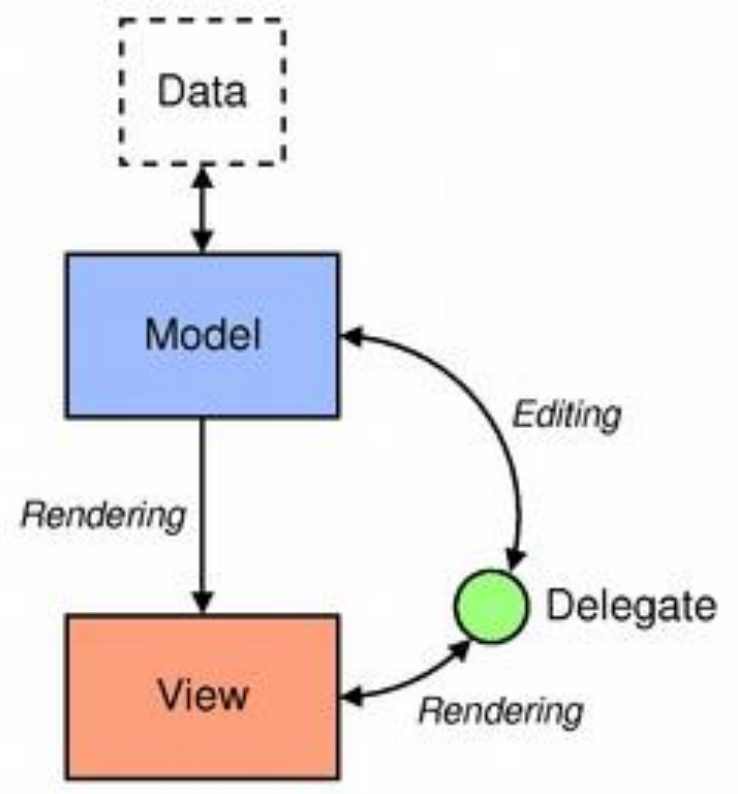
Let’s now focus on Qt …

Model/View Architecture (1/3)

- Built on top of MVC
  - **View** and the **controller** objects are combined
- Still separates
  - the way that the data is stored **(model)** from
  - the way that the data is presented **(view)** to the user
- Developer can
  - Show the same data in different views
  - Implement new views without changing the underlying data structures

Model/View Architecture (2/3)

- The model
  - Communicates with a source of data
  - Provides an interface for the other components in the architecture
- The view
  - Obtains model indexes from the model
  - Uses them to access data
- Delegates
  - Render the items of data
  - It also uses model indexes to communicate



http://doc.qt.io/qt-5/model-view-programming.html

Model/View Architecture (3/3)

- The model/view classes can be separated in
  - Models
  - Views
  - Delegates
- Each of these classes is defined by abstract classes
  - provide common interfaces and def. implementations
- They communicate using **signals** and **slots**

About Signals and Slots

- Signals from the model
  - Inform the view about changes to the data held by the data source
- Signals from the view
  - Provide information about the user's interaction with the items being displayed
- Signals from the delegate
  - Are used during editing to tell the model and view about the state of the editor

Models

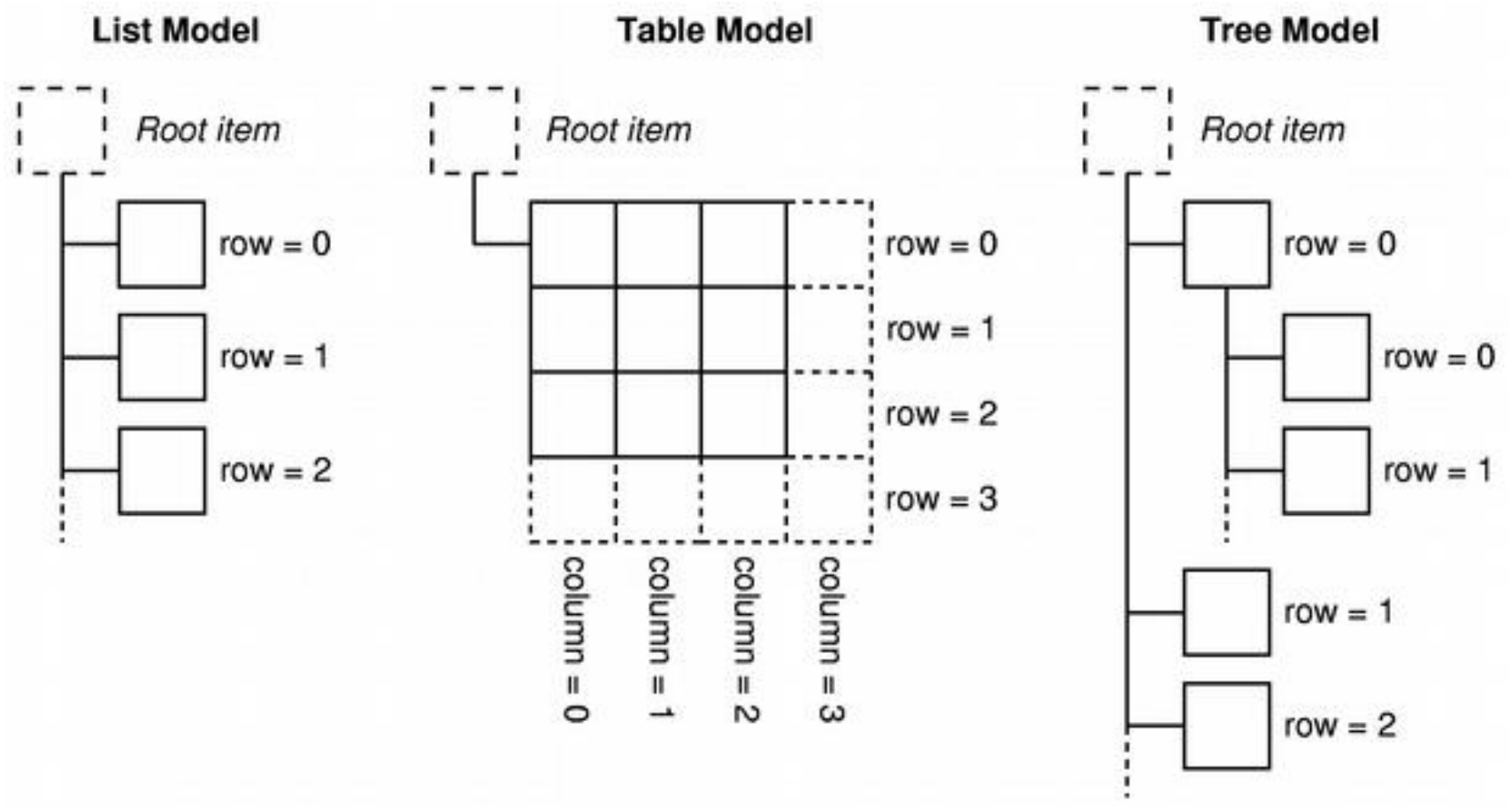
- The data itself **does not** have to be stored in the model
- All item models are based on
  - *QAbstractItemModel* Qt's class
    - Defines an interface that is used by views and delegates to access data
- *QAbstractItemModel* provides a flexible data interface
  - Represents the data in the form of
    - Tables
    - Lists
    - Tree

QAbstractItemModel (1/2)

- Standard interface that item models should use to
  - interoperate with other components in the model/view architecture

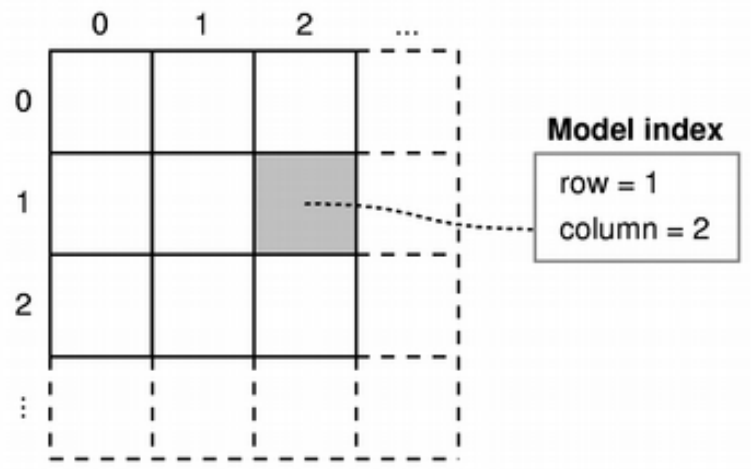
- The underlying data model is exposed to views and delegates as a **hierarchy of tables**
  - If you do not make use of the hierarchy, then the model is a simple table of rows and columns.
  - Each item has a unique index specified by a *QModelIndex*
- Views use this convention to access items of data in the model
  - but this doesn't restrict the way they present the information

QAbstractItemModel (2/2)



http://doc.qt.io/qt-5/model-view-programming.html

Model Index



- **To ensure that the representation of the data is kept separate from the way it is accessed**
  - concept *model index* is introduced
- Each piece of information that can be obtained via a model is addressable by a model index
  - Views and delegates use these indexes to request items of data to display

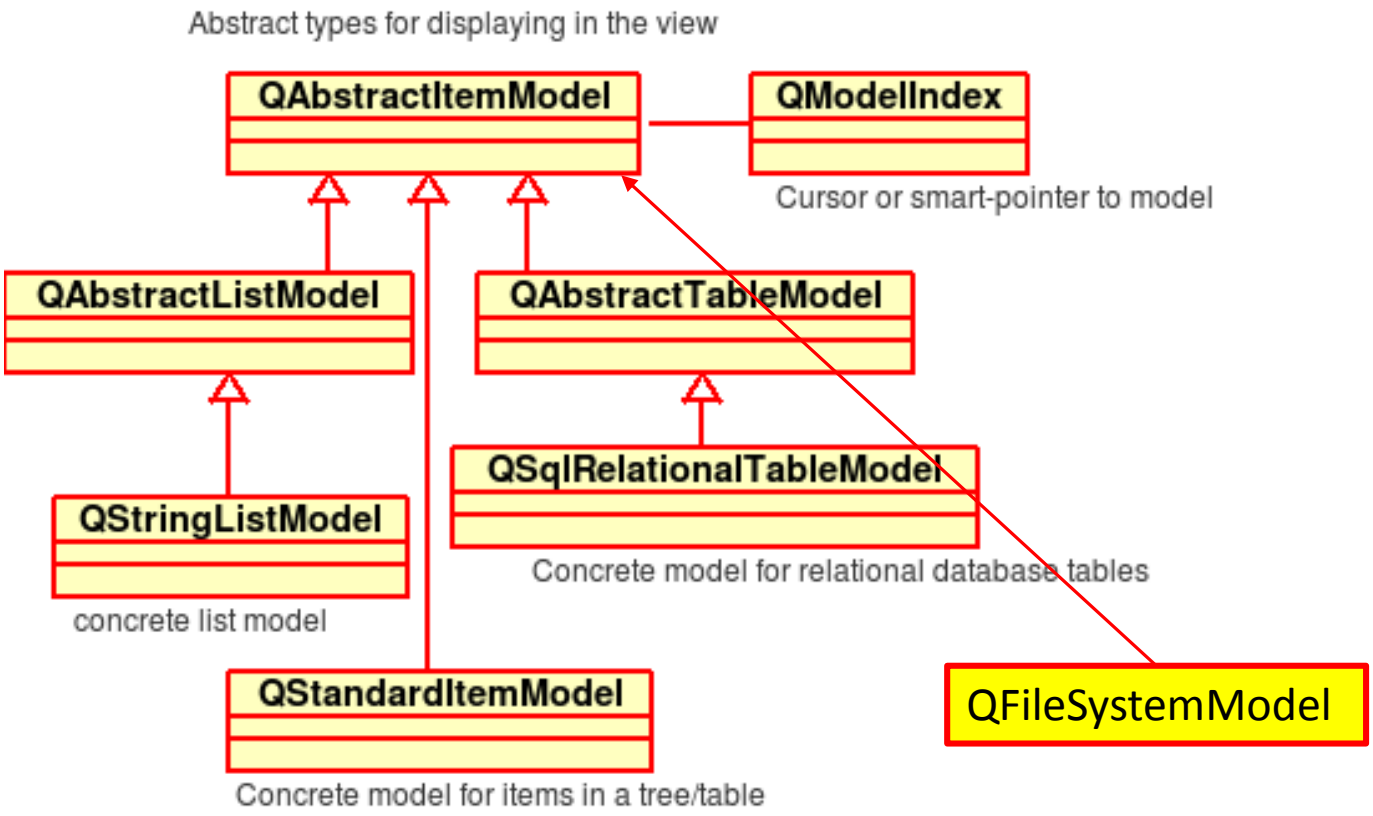
http://doc.qt.io/qt-5/qabstractitemmodel.html#details

Models - Remarks

- In its most basic form a model
  - can be accessed as a simple table in which items are located by their row and column numbers
  - this does not mean that the underlying pieces of data are stored in an array structure
  - the use of row and column numbers is only a convention to allow components to communicate with each other

- The table-like interface to item data provided by models is ideal when using data in a table or list view
  - the row and column number system maps exactly to the way the views display items
  - structures such as tree views require the model to expose a more flexible interface to the items within
    - each item can also be the parent of another table of items
    - in much the same way that a top-level item in a tree view can contain another list of items

Models – Available in Qt



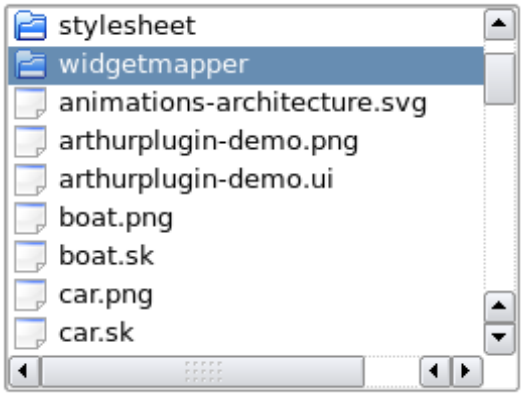
http://www.ics.com/designpatterns/book/qtmodelview.html

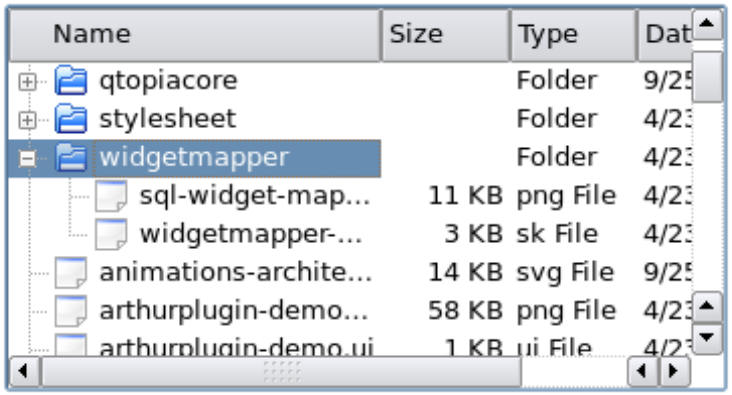
Models – Example

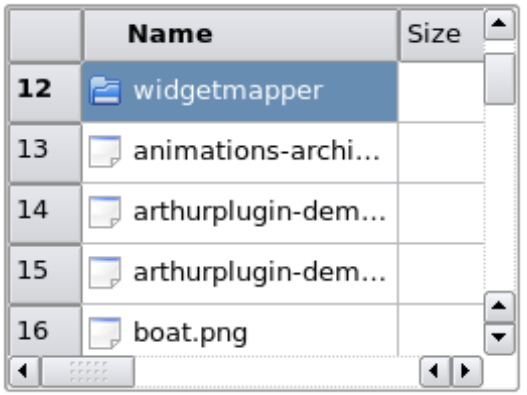
- Using a model
  - Uses *QFileSystemModel*

- chapter13/example13_1

Views

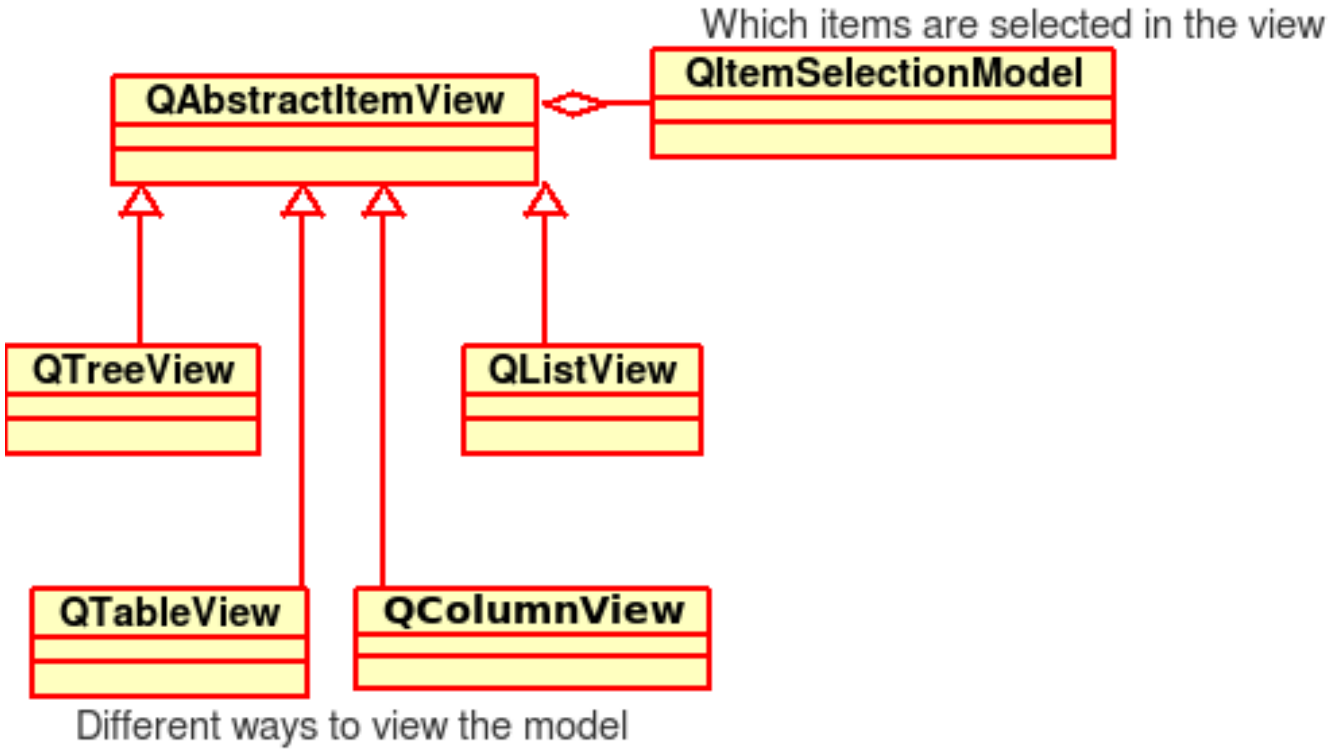






http://doc.qt.io/qt-5/model-view-programming.html

Views – Available in Qt



http://www.ics.com/designpatterns/book/qtmodelview.html

Views – Example

- Multiple views
  - Of the same model
- Populating a model
  - Starting from *QStandardItemModel*

- chapter13/example13_2to13_4

View – Handling Selections

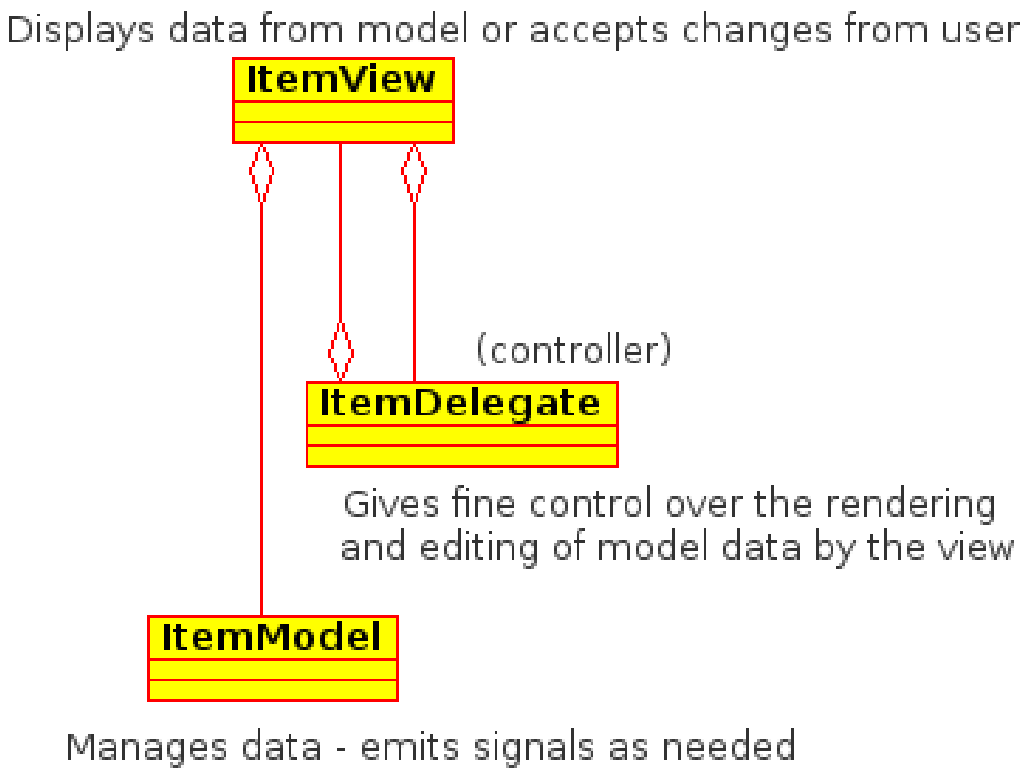
- The mechanism for handling selections of items within view is provided by
  - *QItemSelectionModel* class
- You can personalize it
- You can share selections amongst view

Delegates

- The view is responsible for
  - presentation of model data to the user
  - processing user input
- To allow some **flexibility** in the way this input is obtained
  - interaction is performed by **delegates**

- Delegates
  - provide input capabilities
  - and are responsible for rendering individual items in some views
  - Use *QAbstractItemDelegate* class

Model/View and Delegates

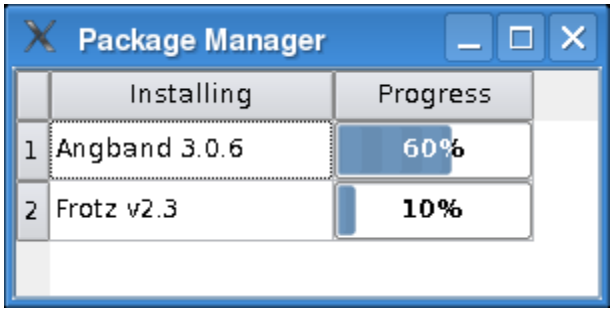


http://www.ics.com/designpatterns/book/delegates.html

Delegates … what? (1/2)

- To render an item in a custom way,
  - you must implement paint() and sizeHint()

- The QItemDelegate class provides default implementations for these functions
  - if you do not need custom rendering, subclass that class instead



http://doc.qt.io/qt-5/qabstractitemdelegate.html#details

Delegates … what? (2/2)

```
void WidgetDelegate::paint(QPainter *painter, const QStyleOptionViewItem &option,
                           const QModelIndex &index) const
{
    if (index.column() == 1) {
        int progress = index.data().toInt();

        QStyleOptionProgressBar progressBarOption;
        progressBarOption.rect = option.rect;
        progressBarOption.minimum = 0;
        progressBarOption.maximum = 100;
        progressBarOption.progress = progress;
        progressBarOption.text = QString::number(progress) + "%";
        progressBarOption.textVisible = true;

        QApplication::style()->drawControl(QStyle::CE_ProgressBar,
                                           &progressBarOption, painter);
    } else
        QStyledItemDelegate::paint(painter, option, index);
```

http://doc.qt.io/qt-5/qabstractitemdelegate.html#details

Delegates – Example

- Extend a delegate
  - *QStyleItemDelegate*

- chapter13/example13_5to13_9

Bonus

Tree Models (s 13.4)

- How to create a concrete model
  - Extending *QAbstractItemModel* class

- chapter13/example13_21to13_22