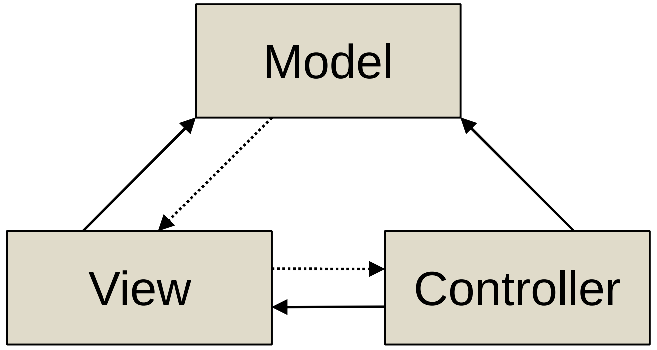
Qt in Education

# The Model View Framework

# The Model View Controller Pattern

- The MVC pattern aims at separating
  - the data (model)
  - the visualization (view)
  - modification (controller)



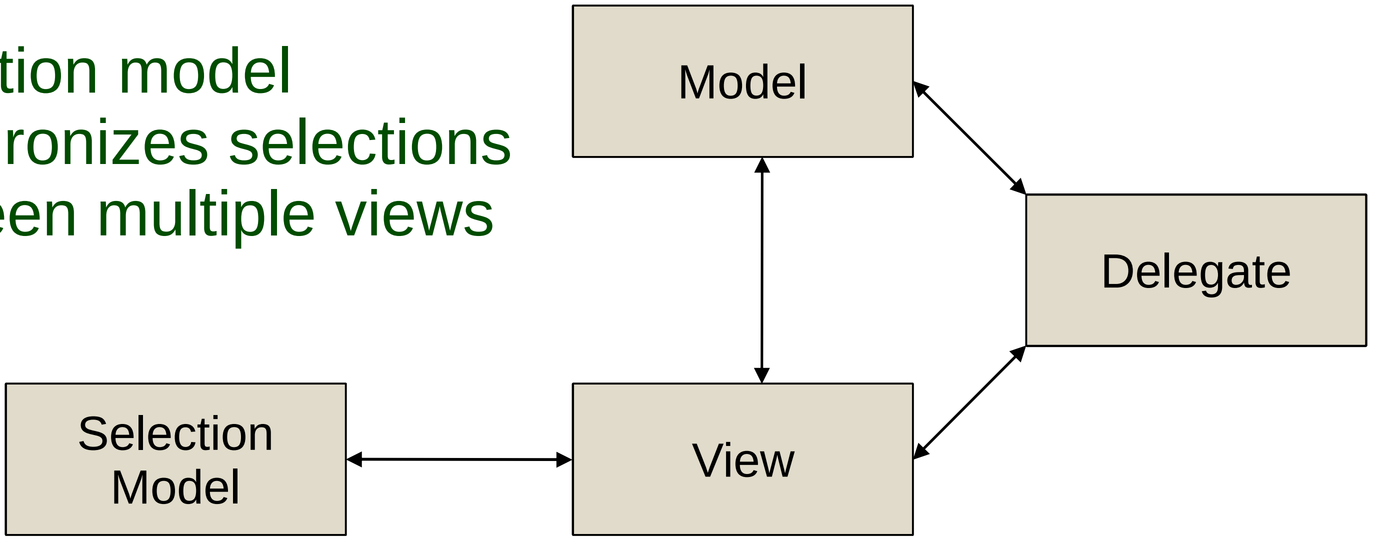
- Provides clear responsibilities for all classes involved

# Why Model View Controller?

- Separates the data from the visualization
  - Avoids data duplication
  - Can show the same data in multiple views
  - Can use the same view for multiple data
- Separates the visualization from the modification
  - Can use application specific actions when altering data
  - The view only needs a single interface for all editing

# Qt's Model View Concept

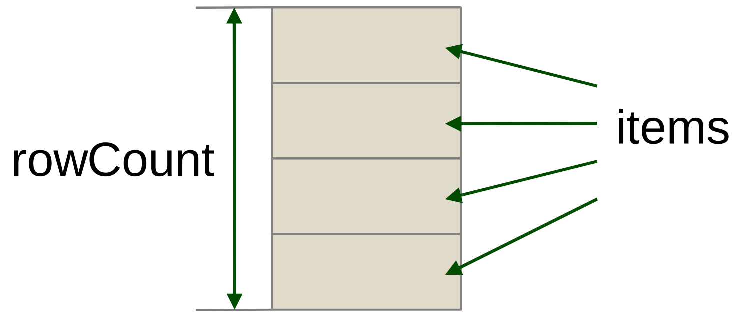
- Qt's Model-View classes are implemented in the Interview framework
  - Model and view
  - Delegate responsible for editing an item for visualization
  - Selection model synchronizes selections between multiple views



# The Model

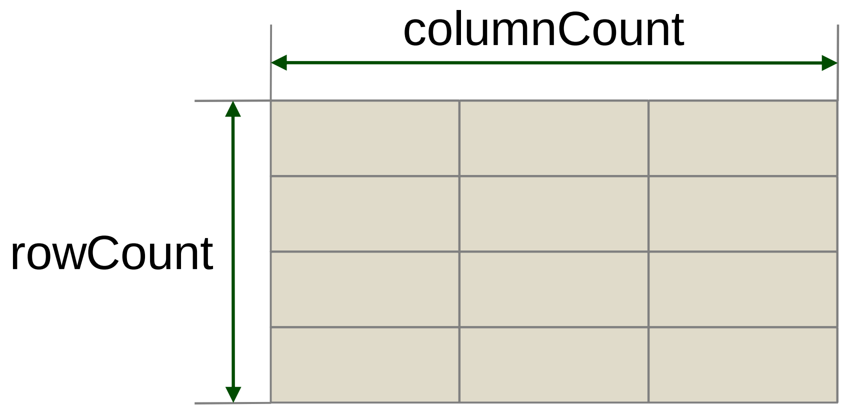
- The abstract model interface class QAbstractItemModel supports
  - Lists – items in one column, multiple rows
  - Tables – items in multiple rows and columns
  - Trees – nested tables

# List models



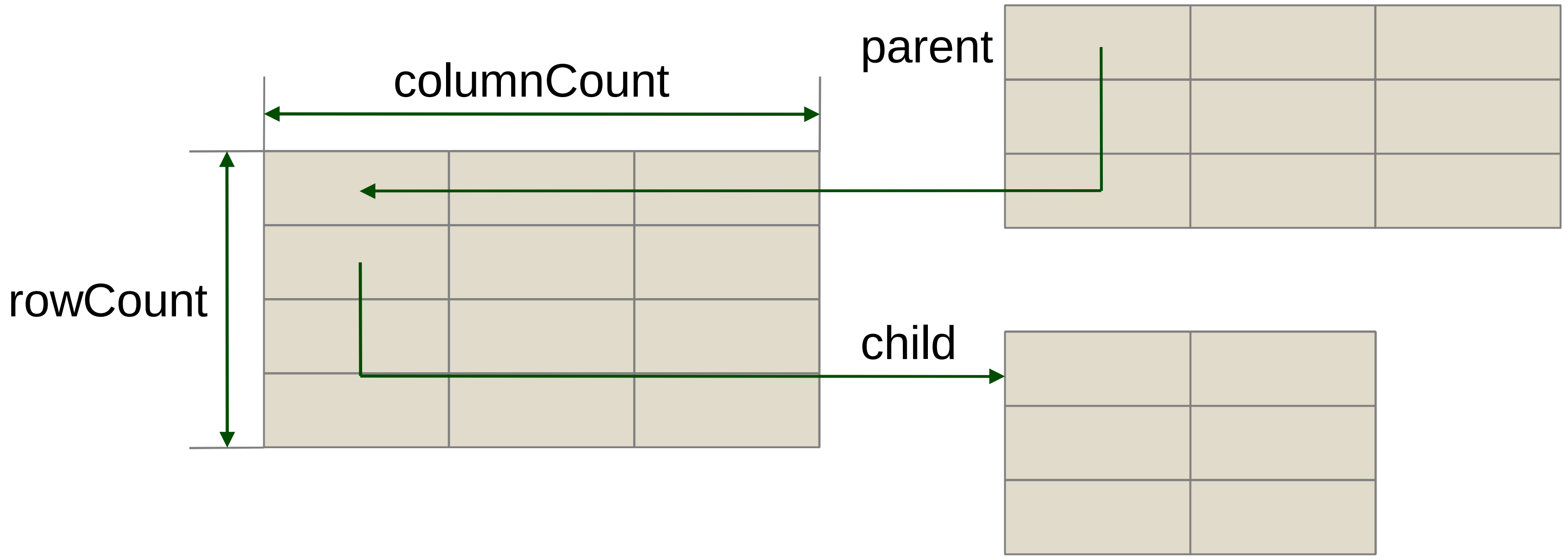
- List models consist of a range of items in a single row
- Each item is addressed by a QModelIndex

# Table models



- A table model places the items in a grid of columns and rows

# Tree models



- A tree model is a table with child tables
- Each sub-table has a QModelIndex as parent
- The top level root has an invalid QModelIndex as parent
- Only items of the first column can be parents

# Data roles

- Each model has a data method used for reading

```
QVariant QAbstractItemModel::data(
    const QModelIndex &index, int role) const
```

- The second argument, role, defaults to Qt::DisplayRole, but there are more roles
  - DecorationRole – for icons, pixmaps, colors, etc
  - EditRole – the data in an editable format
  - FontRole – the font used by the default renderer
  - CheckStateRole – the role to hold the items check state
  - etc

## The QModelIndex

- The model index is used to address individual items of a model
- QAbstractItem model provides the following useful methods
  - index(row, column, parent=QModelIndex())
  - rowCount(parent=QModelIndex())
  - columnCount(parent=QModelIndex())
  - parent(index)
- The QModelIndex provides convenient methods
  - data(role)
  - child(row, column)
  - parent()

## Available models

- In addition to the abstract interface, Qt provides a set of ready to use models
  - QStringListModel – a model exposing a QStringList through the model interface
  - QFileSystemModel – a model exposing file system information (directories and files)
  - QStandardItemModel – a model populated by QStandardItem objects. Can be used to create lists, tables or trees

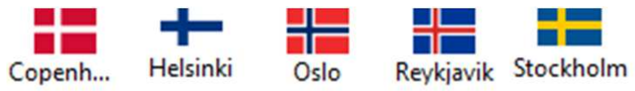
## Available views

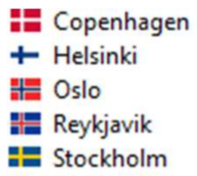
- All views inherit the QAbstractItemView class
- Four views are provided
  - QListView
  - QTableView
  - QTreeView
  - QColumnView

- The QHeaderView widget is used to show headers for rows and columns

## List view

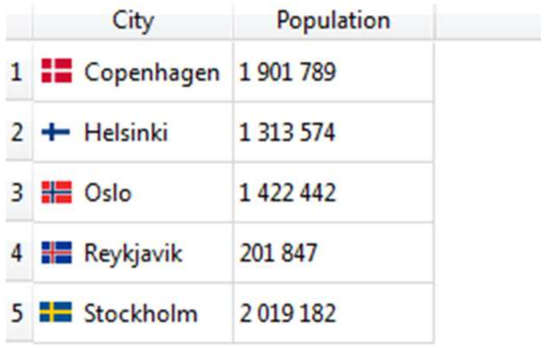
- Shows a single column
  - Use the modelColumn property to select which column
- Provides both IconMode and ListMode





## Table view

- Shows a grid of items



| | City | Population |
|---|---|---|
| 1 | Copenhagen | 1 901 789 |
| 2 | Helsinki | 1 313 574 |
| 3 | Oslo | 1 422 442 |
| 4 | Reykjavik | 201 847 |
| 5 | Stockholm | 2 019 182 |

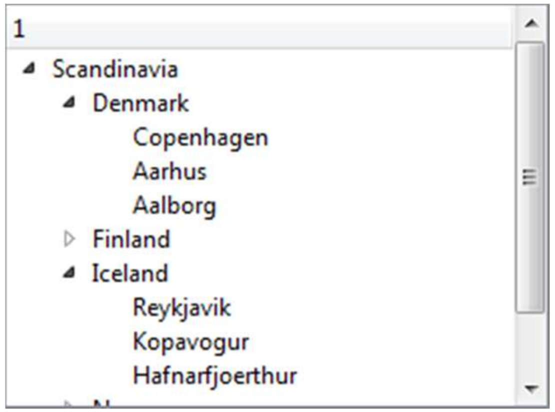
- Use hideRow and hideColumn to hide contents
  - Show it again using showRow and showColumn

## Table View cont'd

- Adapt the grid to the contents using resizeColumnsToContents and resizeRowsToContents
- Access the headers using verticalHeader and horizontalHeader
  - The stretchLastSection property lets the contents fill the width of the widget
  - Headers can be hidden or shown
- Control the scrollbars using the horizontalScrollBarPolicy and verticalScrollBarPolicy properties

# Tree view

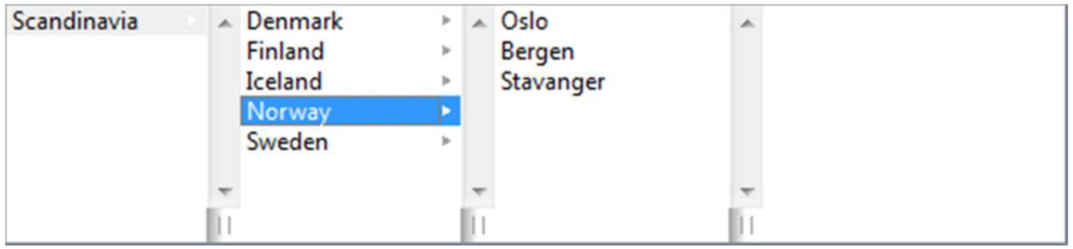
- Shown multi-column trees



- Use setRowHidden and setColumnHidden to hide and show contents
- Use expandAll, expandToDepth and collapseAll to control how much of the tree to show

# Column view

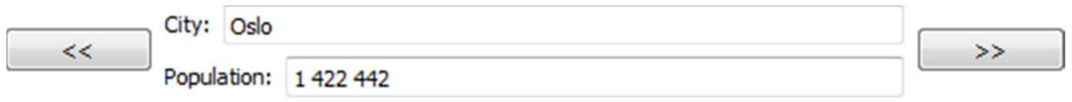
- Shows a tree of lists in separate columns



- Can hold a preview widget in the right-most compartment

# Mapping Data to Widgets

- Using a QDataWidgetMapper, it is possible to map data from a model to widgets



```
QDataWidgetMapper *mapper = new QDataWidgetMapper;

mapper->setModel(model);
mapper->addMapping(cityEdit, 0);
mapper->addMapping(populationEdit, 1);
mapper->toFirst();

connect(nextButton, SIGNAL(clicked()), mapper, SLOT(toNext()));
connect(prevButton, SIGNAL(clicked()), mapper, SLOT(toPrevious()));
```

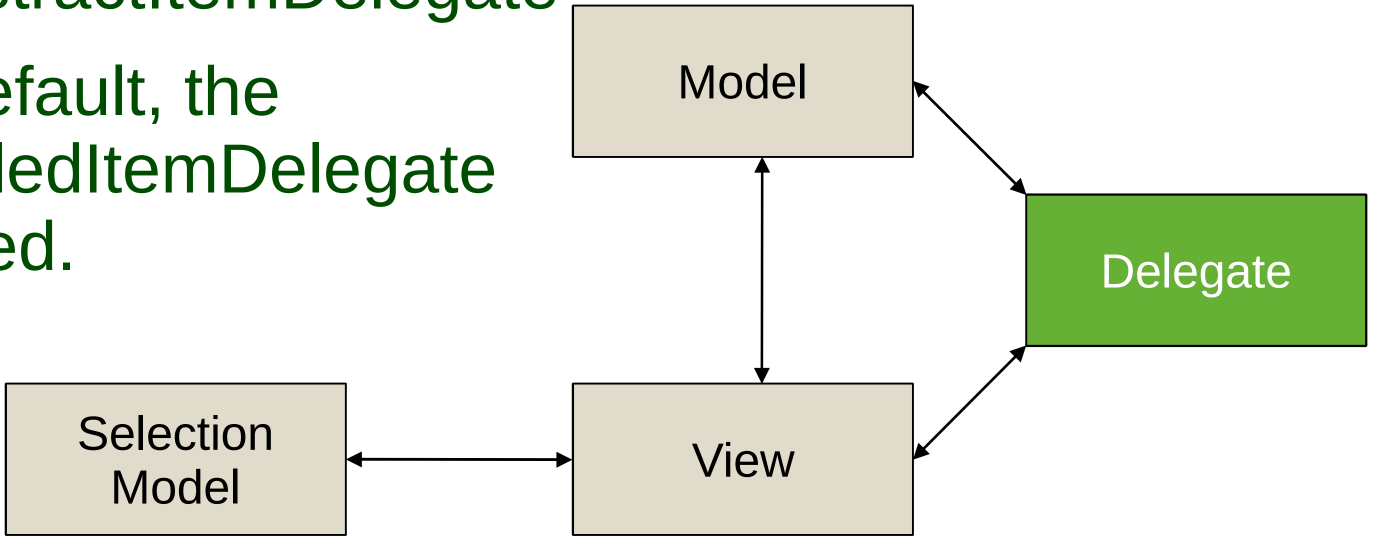
# Widgets with Models

- Sometimes separating the model from the view is too complex
  - No data duplication takes place
  - The model will have to process the data and duplicate it internally
- For these scenarios, the QListWidget, QTableWidget and QTreeWidget exist
  - Uses QListWidgetItem, QTableWidgetItem and QTreeWidgetItem respectively

Break

## The Delegate

- The delegate is responsible for editing and item visualization
  - The view uses and interacts with a delegate
  - All delegates are derived from QAbstractItemDelegate
  - By default, the QStyledItemDelegate is used.



# Delegates and data types

- The QStyledItemDelegate accepts the following data types

| Role | Types |
|---|---|
| CheckStateRole | Qt::CheckState |
| DecorationStyle | QIcon, QPixmap, QImage and QColor |
| DisplayRole | QString (QVariant::toString()) |
| EditRole | |

- The QItemEditorFactor class determines which widget to use for which data type

| Type | Widget |
|---|---|
| bool | QComboBox |
| double | QDoubleSpinBox |
| int / unsigned int | QSpinBox |
| QDate | QDateEdit |
| QDateTime | QDateTimeEdit |
| QPixmap | QLabel |
| QString | QLineEdit |
| QTime | QTimeEdit |

## Custom delegates

- Custom delegates can be implemented to handle painting and/or editing
  - For custom editing but standard painting it is possible to sub-class QItemEditorCreatorBase
- Delegates are assigned to an entire view, columns or rows of views

## Delegate for Painting

- Painting depends on re-implementing the paint and sizeHint methods

```
class BarDelegate : public QStyledItemDelegate
{
    Q_OBJECT
public:
    explicit BarDelegate(int maxRange, QObject *parent = 0);

    void paint(QPainter *painter,
               const QStyleOptionViewItem &option,
               const QModelIndex &index) const;
    QSize sizeHint(const QStyleOptionViewItem &option,
                   const QModelIndex &index) const;

private:
    int m_maxRange;
};
```

# Delegate for Painting

```
BarDelegate::BarDelegate(int maxRange, QObject *parent) :
    QStyledItemDelegate(parent), m_maxRange(maxRange) { }

QSize BarDelegate::sizeHint(const QStyleOptionViewItem &option,
    const QModelIndex &index) const
{
    return QSize(100, 1);
}
```

# Delegate for Painting

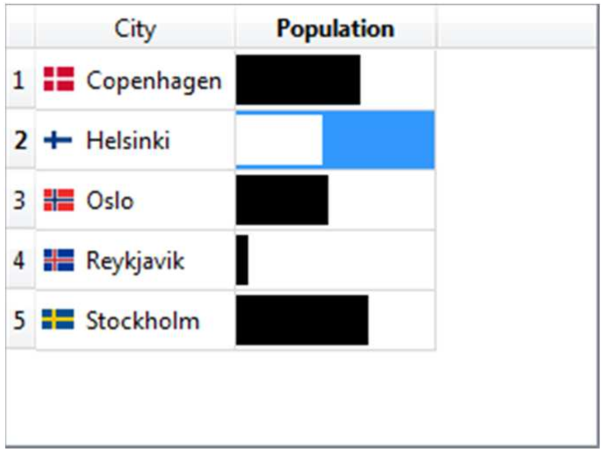
```
void BarDelegate::paint(QPainter *painter,
    const QStyleOptionViewItem &option, const QModelIndex &index) const
{
    if(index.data().canConvert<int>())
    {
        QRect barRect = QRect(option.rect.topLeft(),
            QSize(option.rect.width()*((qreal)index.data().toInt()/(qreal)m_maxRange),
            option.rect.height()));
        barRect.adjust(0, 2, 0, -2);

        if(option.state & QStyle::State_Selected)
        {
            painter->fillRect(option.rect, option.palette.highlight());
            painter->fillRect(barRect, option.palette.highlightedText());
        }
        else
            painter->fillRect(barRect, option.palette.text());
    }
    else
        QStyledItemDelegate::paint(painter, option, index);
}
```

# Using the Delegate

```
tableView->setModel(model);

tableView->setItemDelegateForColumn(1, new BarDelegate(3000000, this));
```



| | City | Population |
|---|---|---|
| 1 | Copenhagen | |
| 2 | Helsinki | |
| 3 | Oslo | |
| 4 | Reykjavik | |
| 5 | Stockholm | |

# Delegates for Editing

- When editing, the view uses the delegate methods createEditor, setEditorData, setModelData and updateEditorGeometry

```
class BarDelegate : public QStyledItemDelegate
{
    Q_OBJECT
public:
...
    QWidget *createEditor(QWidget *parent, const QStyleOptionViewItem &option,
                          const QModelIndex &index ) const;
    void setEditorData(QWidget *editor, const QModelIndex &index) const;
    void updateEditorGeometry(QWidget *editor, const QStyleOptionViewItem &option,
                              const QModelIndex &index) const;
    void setModelData(QWidget *editor, QAbstractItemModel *model,
                      const QModelIndex &index) const;
...
```

- It is common practice to rely on the EditRole and not the DisplayRole for editor data

# Delegates for Editing

```
QWidget *BarDelegate::createEditor(QWidget *parent,
    const QStyleOptionViewItem &option,
    const QModelIndex &index ) const
{
    QSlider *slider = new QSlider(parent);
    slider->setRange(0, m_maxRange);
    slider->setOrientation(Qt::Horizontal);
    slider->setAutoFillBackground(true);

    return slider;
}

void BarDelegate::updateEditorGeometry(QWidget *editor,
    const QStyleOptionViewItem &option,
    const QModelIndex &index) const
{
    QSlider *slider = qobject_cast<QSlider*>(editor);
    if(slider)
        slider->setGeometry(option.rect);
}
```

## Delegates for Editing

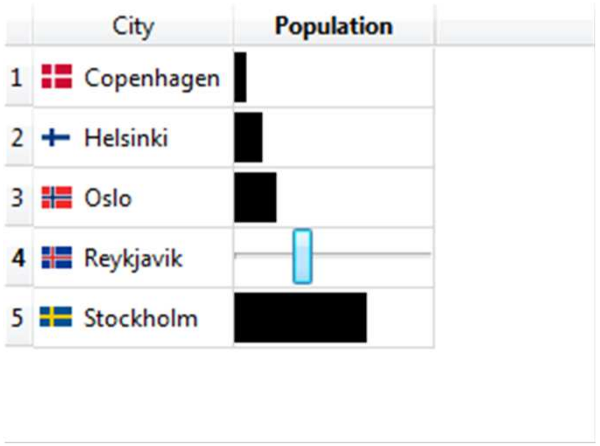
```
void BarDelegate::setEditorData(QWidget *editor, const QModelIndex &index) const
{
    QSlider *slider = qobject_cast<QSlider*>(editor);
    if(slider)
        slider->setValue(index.data(Qt::EditRole).toInt());
}

void BarDelegate::setModelData(QWidget *editor, QAbstractItemModel *model,
    const QModelIndex &index) const
{
    QSlider *slider = qobject_cast<QSlider*>(editor);
    if(slider)
        model->setData(index, slider->value(), Qt::EditRole);
}
```

# Using the Delegate

```
tableView->setModel(model);

tableView->setItemDelegateForColumn(1, new BarDelegate(3000000, this));
```



| | City | Population |
|---|---|---|
| 1 | Copenhagen | |
| 2 | Helsinki | |
| 3 | Oslo | |
| 4 | Reykjavik | |
| 5 | Stockholm | |

# Custom Data Roles

- When working with delegates, it is useful to be able to pass more data between the model and delegate
- It is possible to declare user roles
  - Use Qt::UserRole as first value in enum

```
class CustomRoleModel : public QAbstractListModel
{
    Q_OBJECT
public:
    enum MyTypes { FooRole = Qt::UserRole, BarRole, BazRole };

    ...
```

# Sorting and filtering

- It is possible to sort and filter models using a proxy model
- The QAbstractProxyModel provides
  - mapping between models
  - mapping of selections
- The QSortFilterProxyModel simplifies this by providing interfaces for filtering and sorting
  - The dynamicSortFilter property controls whether the results are to be buffered or generated dynamically

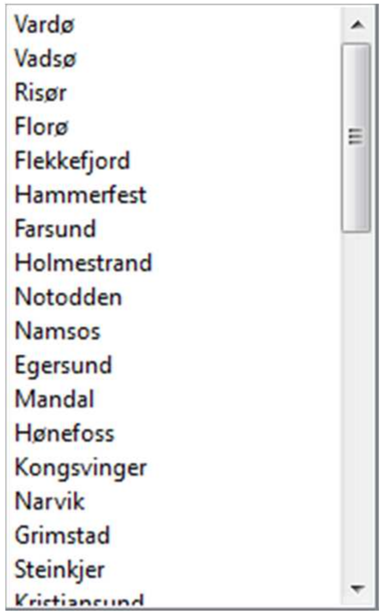
## Sorting

- If the sortingEnabled property is set, clicking the header sorts the contents
  - Applies to QTableView and QTreeView
- By using a QSortFilterProxyModel it is possible to sort on a given column and role
  - sortRole – default DisplayRole
  - sortCaseSensitivity

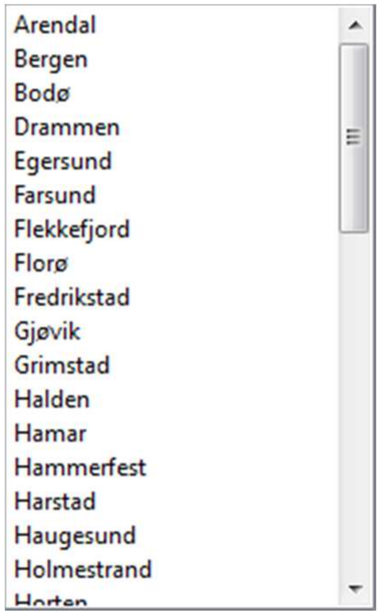
# Sorting Example

```
QSortFilterProxyModel *sortingModel =
    new QSortFilterProxyModel(this);
sortingModel->sort(0, Qt::AscendingOrder);
sortingModel->setDynamicSortFilter(true);
sortingModel->setSourceModel(model);

nonSortedView->setModel(model);
sortedView->setModel(sortingModel);
```



```
Vardø
Vadsø
Risør
Florø
Flekkefjord
Hammerfest
Farsund
Holmestrand
Notodden
Namsos
Egersund
Mandal
Hønefoss
Kongsvinger
Narvik
Grimstad
Steinkjer
Kristiansund
```



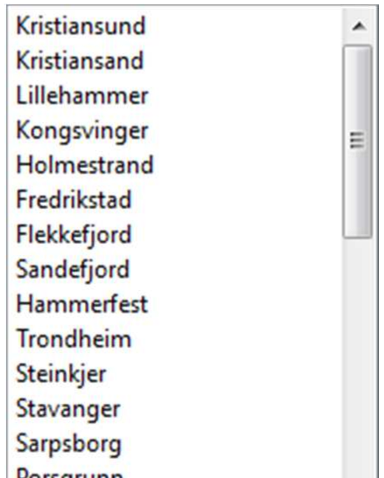
```
Arendal
Bergen
Bodø
Drammen
Egersund
Farsund
Flekkefjord
Florø
Fredrikstad
Gjøvik
Grimstad
Halden
Hamar
Hammerfest
Harstad
Haugesund
Holmestrand
Horten
```

# Custom Sorting

- To implement a more complex sorting algorithm, sub-class and re-implement lessThan method

```
bool MySortProxyModel::lessThan(const QModelIndex &left,
    const QModelIndex &right) const
{
    if(left.data().toString().length() == right.data().toString().length())
        return left.data().toString() < right.data().toString();
    else
        return (left.data().toString().length() < right.data().toString().length());
}
```

```
MySortProxyModel *customSortModel = new MySortProxyModel(this);
customSortModel->sort(0, Qt::DescendingOrder);
customSortModel->setDynamicSortFilter(true);
customSortModel->setSourceModel(model);
customSortedView->setModel(customSortModel);
```



Kristiansund
Kristiansand
Lillehammer
Kongsvinger
Holmestrand
Fredrikstad
Flekkefjord
Sandefjord
Hammerfest
Trondheim
Steinkjer
Stavanger
Sarpsborg

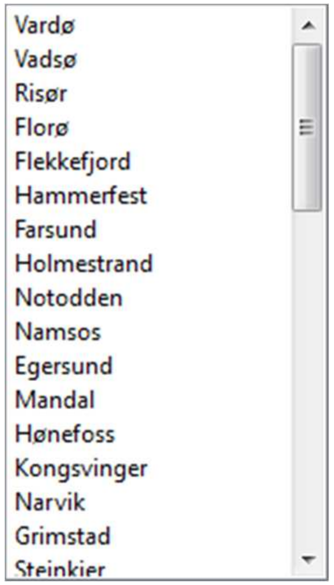
## Filtering

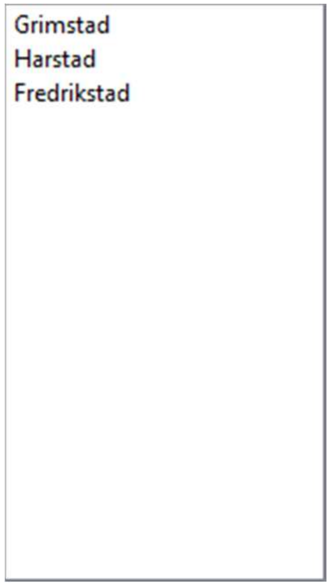
- Filtering makes it possible to reduce the number of rows and columns of a model
  - filterRegExp / filterWildcard / filterFixedString
  - filterCaseSensitivity
  - filterRole
  - filterKeyColumn

# Filter Example

```
QSortFilterProxyModel *filteringModel =
    new QSortFilterProxyModel(this);
filteringModel->setFilterWildcard("*stad*");
filteringModel->setFilterKeyColumn(0);
filteringModel->setDynamicSortFilter(true);
filteringModel->setSourceModel(model);

nonFilteredView->setModel(model);
filteredView->setModel(filteringModel);
```


Vardø
Vadsø
Risør
Florø
Flekkefjord
Hammerfest
Farsund
Holmestrand
Notodden
Namsos
Egersund
Mandal
Hønefoss
Kongsvinger
Narvik
Grimstad
Steinkjer

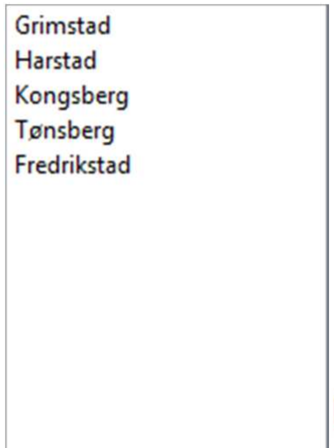

Grimstad
Harstad
Fredrikstad

# Custom Filtering

- To implement more complex filters, sub-class and re-implement the filterAcceptRow and filterAcceptColumn methods

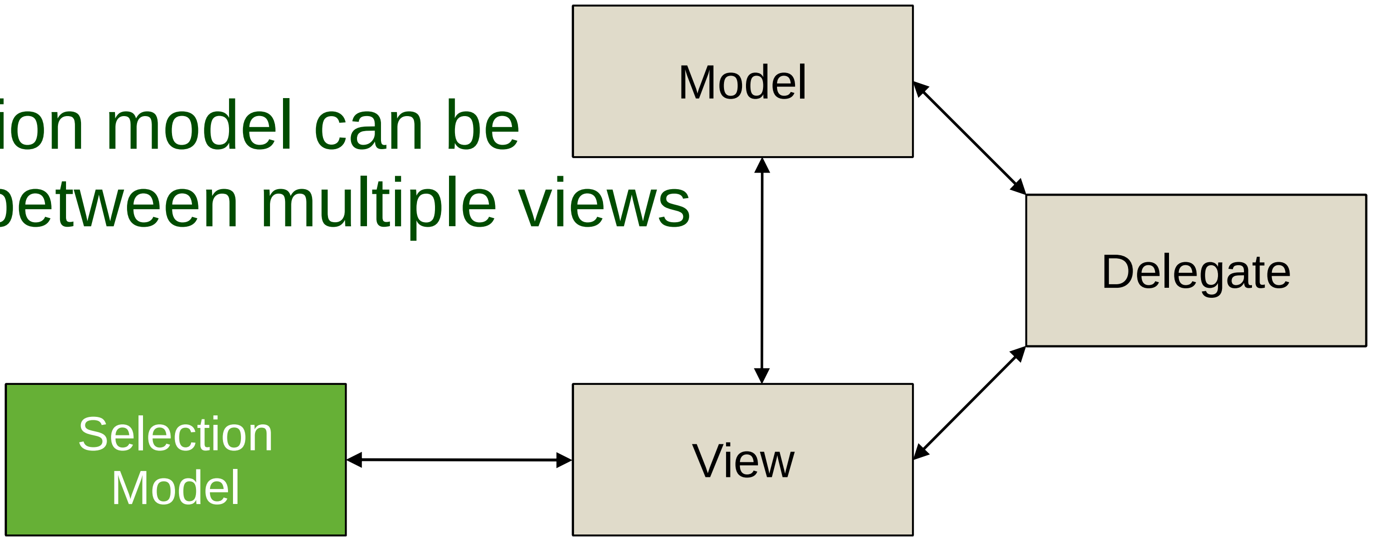
```
bool filterAcceptsRow(int sourceRow, const QModelIndex &sourceParent) const
{
    const QModelIndex &index =
        sourceModel()->index(sourceRow, filterKeyColumn(), sourceParent);
    return index.data().toString().contains("berg") ||
           index.data().toString().contains("stad");
}
```

```
MyFilterProxyModel *customFilterModel = new MyFilterProxyModel(this);
customFilterModel->setFilterKeyColumn(0);
customFilterModel->setDynamicSortFilter(true);
customFilterModel->setSourceModel(model);
customFilteredView->setModel(customFilterModel);
```

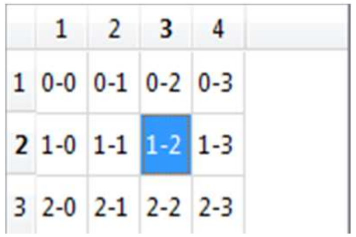

Grimstad
Harstad
Kongsberg
Tønsberg
Fredrikstad

# Working with Selections

- Selections are handled by selection models
- It is possible to tune a view to limit the selection
  - Single items / rows / columns
  - Single selection / contiguous / extended / multi / none
- A selection model can be shared between multiple views

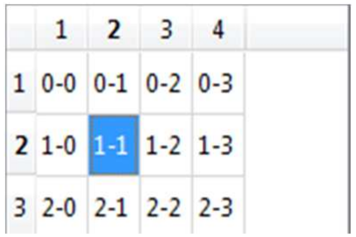


# Selection Behavior and Modes



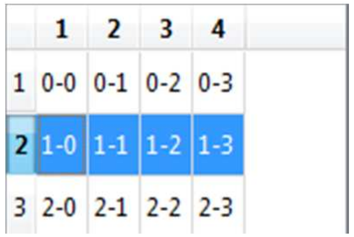
| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0-0 | 0-1 | 0-2 | 0-3 |
| 2 | 1-0 | 1-1 | 1-2 | 1-3 |
| 3 | 2-0 | 2-1 | 2-2 | 2-3 |

```
view->setSelectionBehavior(
    QAbstractItemView::SelectItems);
```



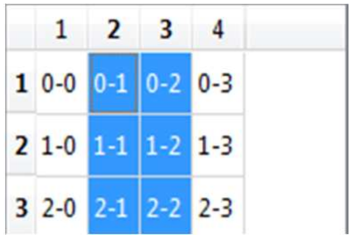
| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0-0 | 0-1 | 0-2 | 0-3 |
| 2 | 1-0 | 1-1 | 1-2 | 1-3 |
| 3 | 2-0 | 2-1 | 2-2 | 2-3 |

```
view->setSelectionMode(
    QAbstractItemView::SingleSelection);
```



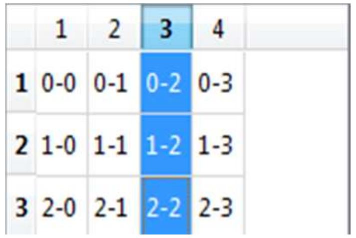
| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0-0 | 0-1 | 0-2 | 0-3 |
| 2 | 1-0 | 1-1 | 1-2 | 1-3 |
| 3 | 2-0 | 2-1 | 2-2 | 2-3 |

```
view->setSelectionBehavior(
    QAbstractItemView::SelectRows);
```



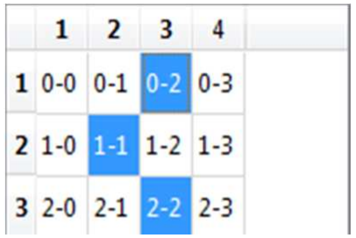
| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0-0 | 0-1 | 0-2 | 0-3 |
| 2 | 1-0 | 1-1 | 1-2 | 1-3 |
| 3 | 2-0 | 2-1 | 2-2 | 2-3 |

```
view->setSelectionMode(
    QAbstractItemView::ContiguousSelection);
```



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0-0 | 0-1 | 0-2 | 0-3 |
| 2 | 1-0 | 1-1 | 1-2 | 1-3 |
| 3 | 2-0 | 2-1 | 2-2 | 2-3 |

```
view->setSelectionBehavior(
    QAbstractItemView::SelectColumns);
```



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0-0 | 0-1 | 0-2 | 0-3 |
| 2 | 1-0 | 1-1 | 1-2 | 1-3 |
| 3 | 2-0 | 2-1 | 2-2 | 2-3 |

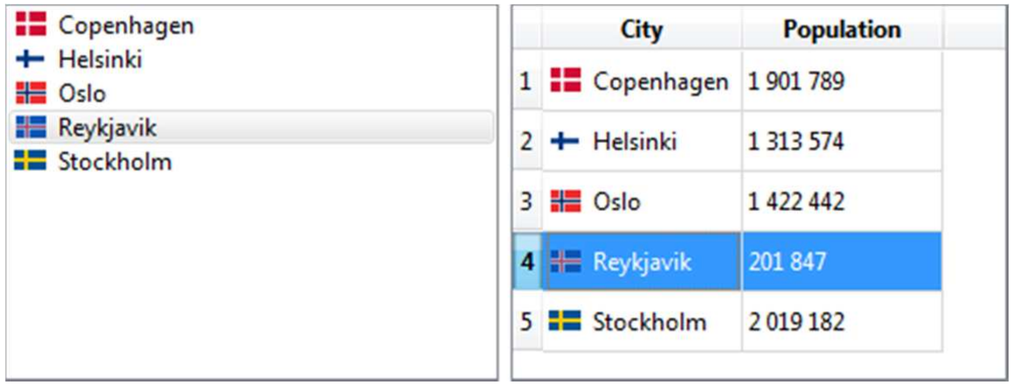
```
view->setSelectionMode(
    QAbstractItemView::ExtendedSelection);
```

# Sharing Selections

- Sharing selections between views, combined with custom views can be a powerful tool

```
listView->setModel(model);
tableView->setModel(model);

listView->setSelectionModel(
    tableView->selectionModel());
```



Copenhagen
Helsinki
Oslo
Reykjavik
Stockholm

| | City | Population |
|---|---|---|
| 1 | Copenhagen | 1 901 789 |
| 2 | Helsinki | 1 313 574 |
| 3 | Oslo | 1 422 442 |
| 4 | Reykjavik | 201 847 |
| 5 | Stockholm | 2 019 182 |

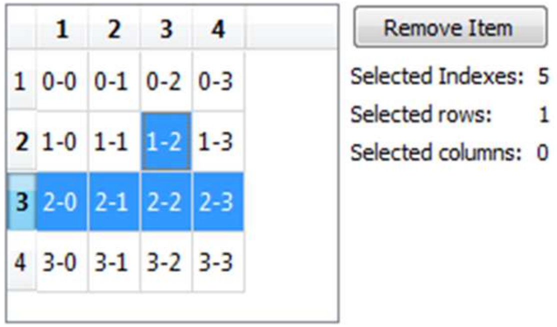
# Reacting to Selection Changes

- Connect to the selection model, not to the view

```
connect(view->selectionModel(), SIGNAL(selectionChanged(QItemSelection,QItemSelection)),
    this, SLOT(updateSelectionStats()));
```

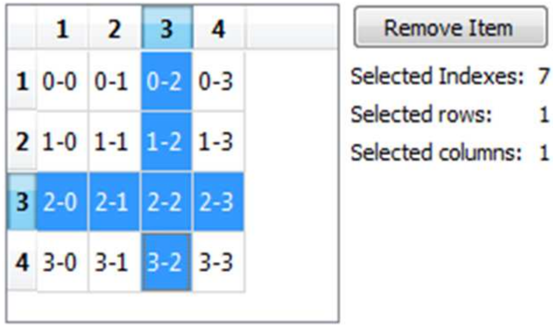
```
void Widget::updateSelectionStats()
{
    indexesLabel->setText(QString::number(view->selectionModel()->selectedIndexes().count()));
    rowsLabel->setText(QString::number(view->selectionModel()->selectedRows().count()));
    columnsLabel->setText(QString::number(view->selectionModel()->selectedColumns().count()));

    removeButton->setEnabled(view->selectionModel()->selectedIndexes().count() > 0);
}
```



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0-0 | 0-1 | 0-2 | 0-3 |
| 2 | 1-0 | 1-1 | 1-2 | 1-3 |
| 3 | 2-0 | 2-1 | 2-2 | 2-3 |
| 4 | 3-0 | 3-1 | 3-2 | 3-3 |

Selected Indexes: 5
Selected rows: 1
Selected columns: 0



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0-0 | 0-1 | 0-2 | 0-3 |
| 2 | 1-0 | 1-1 | 1-2 | 1-3 |
| 3 | 2-0 | 2-1 | 2-2 | 2-3 |
| 4 | 3-0 | 3-1 | 3-2 | 3-3 |

Selected Indexes: 7
Selected rows: 1
Selected columns: 1