Archived:Maintaining square form for a QWidget

Archived: This article is **archived** because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Qt Quick should be used for all UI development on mobile devices. The approach described in this article (using C++ for the Qt app UI) is deprecated.

Overview

This code snippet shows how to create a custom layout manager which can take one item, keep its square form, and center it in the middle of the layout.

Usage

`QSingleItemSquareLayout` works like a `QLayout` should except that it can hold only one item.

Setup code in QMainWindow

```
QWidget* centralWidget = new QWidget;
setCentralWidget(centralWidget);
QSingleItemSquareLayout* mainLayout = new QSingleItemSquareLayout;

QTextEdit* editor = new QTextEdit;
mainLayout->addWidget(editor);

centralWidget->setLayout(mainLayout);
```

Remove and add widget

```
/* Deletes old widget and replaces it with a new one */
void QSISL::changeWidget() {
 QString text = QString("This is widget number %1").arg(this->count);
 this->centralWidget()->layout()->takeAt(0)->widget()->deleteLater();
 this->centralWidget()->layout()->addWidget(new QTextEdit(text));
 ++this->count;
}
```

Header file

```
/*
 * Copyright (c) 2009 Nokia Corporation.
 */

#ifndef QASPECTRATIOLAYOUT_H_
#define QASPECTRATIOLAYOUT_H_

#include <QLayout>
#include <QPointer>
#include <QRect>
```


(C) Copyright Nokia 2014. All rights reserved.

```
#include <QWidgetItem>
#include <QLayoutItem>

/**
 * QSingleItemSquareLayout is a layout which can hold one item,
 * keep its square form and center it.
 */
class QSingleItemSquareLayout : public QLayout
{
 Q_OBJECT

public:
 QSingleItemSquareLayout(QWidget* parent, int spacing =-1);
 QSingleItemSquareLayout(int spacing = -1);
 ~QSingleItemSquareLayout();

 /* Convenience method */
 virtual void add(QLayoutItem* item);
```

Public members

Here are some basic methods you have to implement because you are handling the item references yourself. Explanations for all of these methods can be found in the Qt documentation.

Item manipulation

```
 /* http://doc.trolltech.com/qlayout.html#addItem */
 virtual void addItem(QLayoutItem* item);
 /* http://doc.trolltech.com/qlayout.html#addWidget */
 virtual void addWidget(QWidget* widget);
 /* http://doc.trolltech.com/qlayout.html#takeAt */
 virtual QLayoutItem* takeAt(int index);
 /* http://doc.trolltech.com/qlayout.html#itemAt */
 virtual QLayoutItem* itemAt(int index) const;
 /* http://doc.trolltech.com/qlayout.html#count */
 virtual int count() const;

 /*
  * These are ours since we do have only one item.
  */
 virtual QLayoutItem* replaceItem(QLayoutItem* item);
 virtual QLayoutItem* take();
 virtual bool hasItem() const;
```

Layout and item geometry features

```
 /* http://doc.trolltech.com/qlayout.html#expandingDirections */
 virtual Qt::Orientations expandingDirections() const;

 /*
  * This method contains most of the juice of this article.
  * http://doc.trolltech.com/qlayoutitem.html#setGeometry
  */
 virtual void setGeometry(const QRect& rect);
 /* http://doc.trolltech.com/qlayoutitem.html#geometry */
```


(C) Copyright Nokia 2014. All rights reserved.

```
 virtual QRect geometry();

 /* http://doc.trolltech.com/qlayoutitem.html#sizeHint */
 virtual QSize sizeHint() const;
 /* http://doc.trolltech.com/qlayout.html#minimumSize */
 virtual QSize minimumSize() const;
 /* http://doc.trolltech.com/qlayoutitem.html#hasHeightForWidth */
 virtual bool hasHeightForWidth() const;
```

Private members

```
private:
 /* Saves the last received rect. */
 void setLastReceivedRect(const QRect& rect);
 /* Used to initialize the object. */
 void init(int spacing);
 /* Calculates the maximum size for the item from the assigned size. */
 QSize calculateProperSize(QSize from) const;
 /* Calculates the center location from the assigned size and
  * the items size. */
 QPoint calculateCenterLocation(QSize from, QSize itemSize) const;
 /* Check if two QRects are equal */
 bool areRectsEqual(const QRect& a, const QRect& b) const;
 /* Contains item reference */
 QLayoutItem* item;
 /*
  * Used for caching so we won't do calculations every time
  * setGeometry is called.
  */
 QRect* lastReceivedRect;
 /* Contains geometry */
 QRect* _geometry;

};

#endif /* QASPECTRATIOLAYOUT_H_ */
```

Source file

```
/*
 * Copyright (c) 2009 Nokia Corporation.
 */

#include "qsingleitemsquarelayout.h"

QSingleItemSquareLayout::QSingleItemSquareLayout(QWidget* parent,
                                                 int spacing)
                                                   : QLayout(parent) {
 init(spacing);
}

QSingleItemSquareLayout::QSingleItemSquareLayout(int spacing) {
 init(spacing);
}
```


(C) Copyright Nokia 2014. All rights reserved.

```
QSingleItemSquareLayout::~QSingleItemSquareLayout() {
 delete item;
 delete lastReceivedRect;
 delete _geometry;
}

void QSingleItemSquareLayout::init(int spacing) {
 item = 0;
 lastReceivedRect = new QRect(0, 0, 0, 0);
 _geometry = new QRect(0, 0, 0, 0);
 setSpacing(spacing);
}
```

Public members

Item manipulation

```
/* Adds item if place isn't already taken. */
void QSingleItemSquareLayout::add(QLayoutItem* item) {
 if(!hasItem()) {
  replaceItem(item);
 }
}

/* Adds item if place isn't already taken. */
void QSingleItemSquareLayout::addItem(QLayoutItem* item) {
 if(!hasItem()) {
  replaceItem(item);
 }
}

/* Adds widget if place isn't already taken. */
void QSingleItemSquareLayout::addWidget(QWidget* widget) {
 if(!hasItem()) {
  replaceItem(new QWidgetItem(widget));
 }
}

/* Returns the item pointer and dereferences it here. */
QLayoutItem* QSingleItemSquareLayout::take() {
 QLayoutItem* item = 0;
 if(this->hasItem()) {
  item = this->item;
  this->item = 0;
 }
 return item;
}

/* Returns the item pointer and dereferences it here. */
QLayoutItem* QSingleItemSquareLayout::takeAt(int index) {
 if(index != 0) {
  return 0;
 }
 return this->take();
}
```


(C) Copyright Nokia 2014. All rights reserved.

```
/* Returns the item pointer. */
QLayoutItem* QSingleItemSquareLayout::itemAt(int index) const {
 if(index != 0) {
  return 0;
 }
 if(hasItem()) {
  return this->item;
 }
 return 0;
}

/* Checks if we have an item. */
bool QSingleItemSquareLayout::hasItem() const {
 return this->item != 0;
}

/* Returns the count of items which can be either 0 or 1. */
int QSingleItemSquareLayout::count() const {
 int returnValue = 0;
 if(hasItem()) {
  returnValue = 1;
 }
 return returnValue;
}

/* Replaces the item with the new and returns the old. */
QLayoutItem* QSingleItemSquareLayout::replaceItem(QLayoutItem* item) {
 QLayoutItem* old = 0;
 if(this->hasItem()) {
  old = this->item;
 }
 this->item = item;
 setGeometry(*this->_geometry);
 return old;
}
```

Layout and item geometry features

```
/* Tells which way layout expands. */
Qt::Orientations QSingleItemSquareLayout::expandingDirections() const {
 return Qt::Horizontal | Qt::Vertical;
}

/* Tells which size is preferred. */
QSize QSingleItemSquareLayout::sizeHint() const {
 return this->item->minimumSize();
}

/* Tells minimum size. */
QSize QSingleItemSquareLayout::minimumSize() const {
 return this->item->minimumSize();
}

/*
```


(C) Copyright Nokia 2014. All rights reserved.

```
 * Tells if heightForWidth calculations is handled.
 * It isn't since width isn't enough to calculate
 * proper size.
 */
bool QSingleItemSquareLayout::hasHeightForWidth() const {
 return false;
}

/* Replaces lastReceivedRect. */
void QSingleItemSquareLayout::setLastReceivedRect(const QRect& rect) {
 QRect* oldRect = this->lastReceivedRect;
 this->lastReceivedRect = new QRect(rect.topLeft(), rect.size());
 delete oldRect;
}

/* Returns geometry */
QRect QSingleItemSquareLayout::geometry() {
 return QRect(*this->_geometry);
}

/* Sets geometry to given size. */
void QSingleItemSquareLayout::setGeometry(const QRect& rect) {
 /*
  * We check if the item is set and
  * if size is the same previously received.
  * If either is false nothing is done.
  */
 if(!this->hasItem() ||
    areRectsEqual(*this->lastReceivedRect, rect)) {
  return;
 }
 /* Replace the last received rectangle. */
 setLastReceivedRect(rect);
 /* Calculate proper size for the item relative to the received size. */
 QSize properSize = calculateProperSize(rect.size());
 /* Calculate center location in the rect and with item size. */
 QPoint properLocation = calculateCenterLocation(rect.size(), properSize);
 /* Set items geometry */
 this->item->setGeometry(QRect(properLocation, properSize));
 QRect* oldRect = this->_geometry;
 /* Cache the calculated geometry. */
 this->_geometry = new QRect(properLocation, properSize);
 delete oldRect;
 /* Super classes setGeometry */
 QLayout::setGeometry(*this->_geometry);
}
```

Private members

Layout calculations

```
/* Takes the shortest side and creates QSize
 * with the shortest side as width and height. */
QSize QSingleItemSquareLayout::calculateProperSize(QSize from) const {
 QSize properSize;
 if(from.height() < from.width()) {
```


(C) Copyright Nokia 2014. All rights reserved.

```
  properSize.setHeight(from.height() - this->margin());
  properSize.setWidth(from.height() - this->margin());
 }
 else {
  properSize.setWidth(from.width() - this->margin());
  properSize.setHeight(from.width() - this->margin());
 }
 return properSize;
}

/* Calculates center location from the given height and width for item size. */
QPoint QSingleItemSquareLayout::calculateCenterLocation(QSize from,
                                                        QSize itemSize) const {
 QPoint centerLocation;
 if(from.width() - from.width()/2 - itemSize.width()/2 > 0) {
  centerLocation.setX(from.width() -
                      from.width()/2 -
                      itemSize.width()/2);
 }
 if(from.height() - from.height()/2 - itemSize.height()/2 > 0) {
  centerLocation.setY(from.height() -
                      from.height()/2 -
                      itemSize.height()/2);
 }
 return centerLocation;
}

/* Compares if two QRects are equal. */
bool QSingleItemSquareLayout::areRectsEqual(const QRect& a,
                                            const QRect& b) const {
 bool result = false;
 if(a.x() == b.x() &&
    a.y() == b.y() &&
    a.height() == b.height() &&
    a.width() == b.width()) {
  result = true;
 }
 return result;
}
```

Postconditions

You know how to maintain square form for a widget with a custom layout manager and know the basics of implementing a layout manager.

Supplementary material

- You can test the QSingleItemSquareLayout manager with a test application. The application is available for download at Media:QSISL example.zip.

See also

- See Archived:Implementing a layout manager in Qt for more general example of creating layout manager.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.