Thema, Ziele: qmake mit .pro Datei, QObject

Aufgabe 1: Counter mit qmake

Erstellen Sie ein qmake Projekt (*Counter.pro*) mit der Counter Klasse vom letzten Praktikum und dessen Test (siehe Vorlage/Counter Ordner). Verwenden Sie dazu noch nicht den Qt Creator.

Aufgabe 2: Stack App mit Stack Library und qmake

Nutzen Sie die Stack Library vom letzten Praktikum (siehe Vorlagen Ordner) mit Hilfe von qmake. Erstellen Sie wiederum eine Testapplikation *Stack*.

Hinweis: Das Einbinden kann bei qmake respektive dem .pro File wie folgt aussehen:

GCC-Flags	.pro qmake Flags
-I*path*	INCLUDEPATH += *path*
-L*path* -l*libName*	LIBS += -Lpath -llibName

Aufgabe 3: QObject auf Heap

Untersuchen Sie das Verhalten von QObjects auf dem Heap.

Vorgehen:

1. Instanziieren Sie zwei QObject Objekte *o1* und *o2* auf dem Heap. Achten Sie dabei darauf, dass *o2* ein Child von *o1* wird.
2. Löschen Sie *o1* vom Heap

Was passiert mit *o2*? Überprüfen Sie das Verhalten mittels ableiten von QObject mit der Klasse *Test*. Schreiben Sie eine entsprechende Konsolenausgabe im D'tor der *Test* Klasse. Nun sollen anstatt den QObjects *o1* und *o2* die Objekte von *Test t1* und *t2* verwendet werden.

Hinweis:

Die Ausgabe kann via *qDebug* (#include <QDebug>)wie folgt aussehen:

```
qDebug() << "Test D'tor called" << this->objectName() << " object called";
```

Dabei wird das Attribut objectName gentzt, dass nach dem Erstellen des Objekts gesetzt werden kann:

```
Test* t1 = new Test;

t1->setObjectName("t1");
```

Aufgabe 4: QObject auf Stack

Wiederholen Sie Aufgabe 4 mit dem Unterschied, dass die Objekte nun auf dem Stack und nicht auf dem Heap zu liegen kommen sollen.

Wieso ist die *setParent*-Methode der *QObject* Klasse bei Objekten auf dem Stack gefährlich?