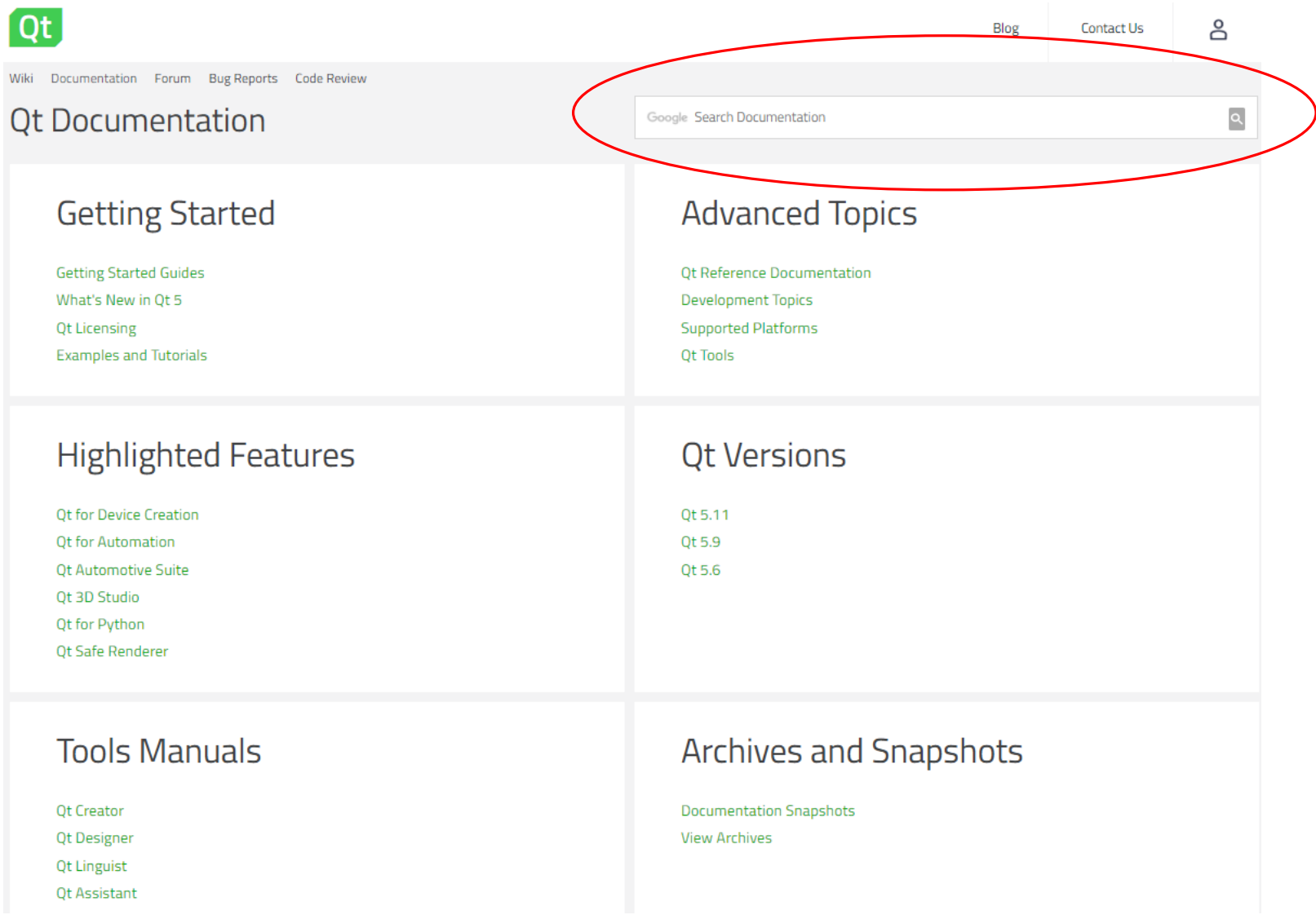
Olio-ohjelmointi: Qt –luokkakirjaston käyttö

- Avaa selaimeen sivu www.qt.io ja siirry sivun loppuun ja klikkaa otsikon **Developers** alapuolella olevaa linkkiä **Documentation**
- Selaimeen aukeaa oikealla olevan kuvan mukainen sivu. Tämän sivun kautta on hyvä tutustua Qt kehitysympäristöön.
- Klikkaa otsikon **Qt Versions** alla olevaa uusinta Qt Version linkkiä
- Klikkaa avautuvalta sivulla linkkiä **All Qt C++ Classes**. Pelkästään selaamalla läpi Qt –luokkakirjastoa näkee, kuinka moneen eri asiaan on kirjastossa valmiita luokkia. Tämä helpottaa ohjelmointia, kun kaikkea ei tarvitse tehdä itse alusta alkaen.
- On tärkeää opetella käyttämään yhtä luokkakirjastoa, koska muiden vastaavien käyttö on sen jälkeen helpompi oppia.



Olio-ohjelmointi: Qt –luokkakirjaston käyttö

- Tee uusi projekti joka on tyyppiä **Application->Qt Console Application** (anna projektille nimeksi **MyQtConsoleApp**)
- Rakenna projektin aikaisemmin opitun mukaisesti.
- Avaa projektin puurakenteesta tiedosto **MyQtConsoleApp.pro**

  - Rivi **QT -= gui** tarkoittaa, että paketti gui (=graphical user interface) on pois projektista, koska tehdään **console** sovellus, eikä graafista sovellusta.
  - Riveille **SOURCES** ja **HEADERS** tulee automaattisesti kaikki kooditiedostot (.h ja .cpp), jotka projektiin kuuluvat, kun niitä aletaan projektiin lisäämään.

- Voit sulkea tiedoston **MyQtConsoleApp.pro**

Olio-ohjelmointi: Qt –luokkakirjaston käyttö

- Avaa **main()** funktio

  - Mistä luokasta luodaan olio, mikä on olion nimi ja miltä muistialueelta oliolle varataan tilaa?

- Siirry takaisin selaimeen ja kirjoita hakukenttään sana **QCoreApplication** ja paina **Enter** painiketta

- Avaa hakulistalta linkki (**QCoreApplication Class | Qt Core …**) jolloin saat esille tietoa luokasta.

  - Minkä otsikkotiedoston (header) **QCoreApplication** vaatii käännösvaiheessa, mihin **Qt** pakettiin **QCoreApplication** kuuluu ja minkä luokan **QCoreApplication** perii (Inherits)?

- Klikkaa ensimmäisessä kappaleessa olevaa linkkiä **more** ja selvitä yleisellä tasolla mikä on luokan tarkoitus.

- **Event Loop** (sanomajono) käsite käydään läpi seuraavilla sivuilla.

Olio-ohjelmointi: Tapahtumien (=Events) käsittely sovelluksessa

- Kun ohjelma/järjestelmä on käynnissä (esim. pankkiautomaatti), niin sen täytyy pystyä reagoimaan erilaisiin ulkoisiin/sisäisiin tapahtumiin, jotka vaikuttavat ohjelman/järjestelmän toimintaan.
- Nämä tapahtumat ovat sellaisia, että niiden tapahtuma-ajasta järjestelmässä ei aina voida tietää välttämättä etukäteen.
- Kun näitä eri tapahtumia tulee ohjelman käsiteltäväksi, niin ne on saatava jotenkin **jonoon**, josta ne voidaan ottaa käsittelyyn yksitellen.
- Ulkoisia tapahtumia aiheuttavat ohjelman kanssa kommunikoiva ympäristö (käyttäjät, tietoverkko, pilvipalvelut, tietokannat, anturit, mobiililaitteet yms.)

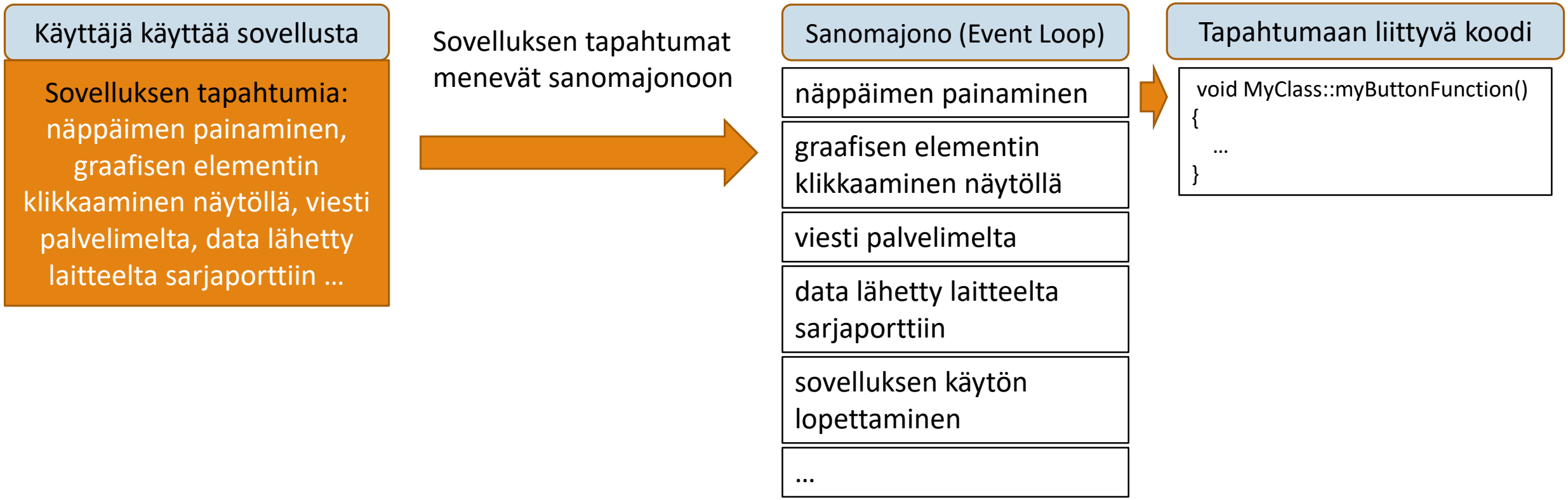
- Pankkiautomaatin kanssa ulkoisen ympäristön tapahtumia ovat esim. seuraavat:
  - Käyttäjä käyttää automaatin näppäimistöä tiedon syötössä.
  - Käyttäjä käyttää automaatin kosketusnäyttöä ohjelman toimintoja valitessaan.
  - Ohjelma saa kortinlukulaitteelta tiedon/signaalin, että kortti on laitettu lukulaitteeseen.
  - Kuittikirjoitin tulostaa kuittipaperille käyttäjän valinnan mukaan.
  - Ohjelma on yhteydessä pankin tietokantaan tietoverkon välityksellä (esim. haku- ja päivitystoiminnot).

- Kaikki nämä toiminnot aiheuttavat tapahtumia/signaaleja rakennettuun ohjelmaan, johon ohjelman on pystyttävä reagoimaan.
- Tähän tarkoitukseen on olemassa eri ohjelmointikielissä tapahtumankäsittelyrakenne (Event Handling). Tapahtumankäsittely hoidetaan käyttöjärjestelmän ja ohjelmistokehitysympäristön yhteistyöllä.

- Qt ohjelmistokehitysympäristössä tapahtumienkäsittely on hoidettu sanomajonon avulla. Siitä seuraavilla sivuilla.

Olio-ohjelmointi: Event Loop, sanomajono

- Ohjelmassa tarvitaan sanomajono, jos ohjelman täytyy pystyä ottamaan signaaleja vastaan järjestelmän eri osilta mihin aikaan tahansa.
- **QCoreApplication** luokka tarjoaa tapahtumankäsittelyä varten sanomajonon (Event Loop) sovelluksille, joissa ei ole graafista käyttöliittymää.
- Graafisille sovelluksille on oma luokkarakenne, mutta käytännössä sama sanomajonotoiminnallisuus. Tähän palataan myöhemmin.
- Sanomajono toimii alla olevan mukaisesti.



Olio-ohjelmointi: Event Loop, sanomajono

- Qt:n sivuilla on luokasta **QCoreApplication** kerrottu seuraavaa liittyen sanomajonoon:

> *The QCoreApplication class provides an event loop for Qt applications without UI. For non-GUI application that uses Qt, there should be exactly one QCoreApplication object. QCoreApplication contains the main event loop, where all events from the operating system (e.g., timer and network events) and other sources are processed and dispatched. It also handles the application's initialization and finalization, as well as system-wide and application-wide settings.*
>
> *The event loop is started with a call to exec().*

- Mikä on funktion **exec()** tarkoitus? Mikä viesti/funktiokutsu aiheuttaa sen, että ohjelman sanomajono suljetaan?

- Muuta koodirivi **return a.exec();** alla olevan mukaiseksi, niin funktion toiminto tulee selkeämmin esille.

```
int execReturnValue;
execReturnValue = a.exec(); // jaadaan sanomajonoon odottamaan quit viestiä, jolla poistutaan sanomajonosta.
return execReturnValue;
```

Olio-ohjelmointi: Qt –luokkakirjasto, SIGNAL/SLOT -rakenne

- Qt-ohjelmistokehitysympäristössä voidaan lähettää sanomajonoon viestejä signaalien avulla.
- Signaalit voidaan liittää Qt-ohjelmistokehitysympäristössä erilaisiin tapahtumiin. Esimerkiksi: näppäimen painaminen, graafisen elementin klikkaaminen näytöllä, viesti palvelimelta, dataa sarjaportissa, sovelluksen käytön lopettaminen jne.
- Ohjelmoijat voivat itse luoda signaaleja tai sitten voidaan käyttää Qt –luokkien omia signaaleja.
- Qt -ohjelmistokehitysympäristössä signaali voidaan kytkeä luokan jäsenfunktioon (Qt:ssa käytetään termiä SLOT tällaisista jäsenfunktioista), joka suoritetaan kun signaali tapahtuu ohjelmassa. Tästä tulee käsite **SIGNAL/SLOT** Qt:ssa.
- **SLOT** tyyppiset jäsenfunktiot esitellään luokan rakenteessa osassa **public slot** tai **private slot**. Ohjelmassa voidaan käyttää myös Qt luokkien jäsenfunktioita, joita kutsutaan kun signaali suoritetaan (esimerkkinä **quit()** jäsenfunktio myöhemmin).

Olio-ohjelmointi: Qt –luokkakirjasto, SIGNAL/SLOT -rakenne

- Lisää ohjelmaan luokka **MyClass**. HUOM! Kun lisäät luokkaa, niin valitse kohtaan **Base Class** luokka **QObject** ja varmista, että luokkaa lisättäessä ikkunassa **Define Class** on rasti kohdassa **Include QObject**. Edellä tehdyillä asetuksilla luokka **MyClass** laitetaan perimään Qt luokkahierarkian kantaluokka **QObject**.

- Kirjoita selaimessa hakukenttään sana **QObject,** valitse hakulistalta **QObject Class | Qt Core …** ja selvitä luokan tarkoitus yleisellä tasolla.

- Avaa luokan **MyClass** rakenne (myclass.h) editoriin ja käy läpi luokan rakenteen koodi alustavasti. Luokassa on uusia osia, joita käydään läpi tulevilla sivuilla.

- Suorita build ja aja ohjelmaa.

- Koska ohjelmassa ei ole mitään tapahtumia eikä mitään signaaleja, niin ohjelman pitäisi päättyä normaalisti!

  Mutta konsoli-ikkunaan ei kuitenkaan tule tuttua tekstiä ***Press <RETURN> to close this window...***

- Miksi ohjelma ei päätykkään normaalisti?

- Sammuta ohjelma klikkaamalla konsoli-ikkunan oikeassa yläkulmassa olevaa rastia, ja klikkaa tämän jälkeen Qt Creatorissa alhaalla olevaa kohtaa **3 Application Output**, joka ilmoittaa että **”…MyQtConsoleApp.exe crashed”**
  Eli ohjelma kaatui, koska sanomajono ei saanut prosessin lopettamiskäskyä!

Olio-ohjelmointi: Qt –luokkakirjasto, SIGNAL/SLOT –rakenne

- Koska projekti **MyQtConsoleApp** on sen tyyppinen projekti, että siinä käytetään luokkaa **QCoreApplication** ja se taas vaatii sanomajonon käytön, niin siksi ohjelma kaatuu jos sovellusta/prosessia ei lopeteta/ajeta alas oikein.

- Siirry selaimeen ja etsi luokan **QCoreApplication** esittelystä osio **Public Slots**. Tästä osiosta löytyy jäsenfunktio **quit()**. Siirry jäsenfunktion kuvaukseen ja käy se läpi yleisellä tasolla.

- Jotta konsoli-ikkunassa suoritettava sanomajonopohjaisen ohjelman käyttö lopetetaan oikein, pitää johonkin ohjelman signaaliin liittää luokan **QCoreApplication** jäsenfunktion **quit()** suorittaminen. Kun tämä **quit()** jäsenfunktio tulee sanomajonossa suoritukseen, niin ohjelma lopetetaan oikein ja **Application Output** ikkunaan tulee viesti **“…MyQtReadFileApp.exe exited with code 0”**.

- Avaa luokan **MyClass** rakenne **Qt Creator** editoriin. Seuraavalla sivulla tutustutaan luokkaan ja lisätään siihen koodia.

```
#ifndef MYCLASS_H
#define MYCLASS_H

#include <QObject>

#include <iostream>
using namespace std;

class MyClass : public QObject
{
   Q_OBJECT
public:
    MyClass(QObject *parent = nullptr);
   ~MyClass();
   void raiseFinishProgramSignal();

signals:
   void finishProgram();

public slots:
};

#endif // MYCLASS_H
```

Luokan rakenne ja ohjelman toiminnot

- Luokka **MyClass** perii luokan **QObject**

- **Q_OBJECT** on makro, joka tarvitaan aina kun halutaan käyttää ohjelmassa sanomajonoa, lähettää signaaleja ja käyttää **SIGNAL/SLOT** rakennetta.
- **explicit** sana muodostinfunktiossa käydään läpi myöhemmin. Sen voi halutessaan tässä vaiheessa poistaa.

- Lisää luokkaan tuhoajafunktio ja jäsenfunktio **void raiseFinishProgramSignal();**
- Lisää osaan **signals** rivi **void finishProgram();** Eli luodaan oma signaali, jonka kutsu aiheuttaa jäsenfunktion **quit()** kutsumisen ja ohjelman sanomajonosta poistumisen ja ohjelman/prosessin lopettamisen oikein.
- **public slots** osaan tutustutaan myöhemmin.

- Lisää muodostinfunktioon koodirivi **cout << "Olio objectMyClass luotu"<< endl;**
- Lisää tuhoajafunktioon koodirivi **cout << "Olio objectMyClass tuhottu"<< endl;**

- Lisää funktioon **raiseFinishProgramSignal()** alla olevat koodirivit

cout << "Signaali finishProgram() ylos" << endl;
***emit** finishProgram(); // **emit** kaskylla nostetaan signaali ylos*

- Seuraavalla sivulla käydään läpi miten signaalia **finishProgram()** käsitellään tässä ohjelmassa.
- Ennen kuin jatkat niin tarkista, että luokan rakenne vastaa vasemmalla olevaa koodia.

main() funktion toiminnot

- Kirjoita **main()** funktio vasemmalla olevan koodin mukaiseksi
- Signaalin **finishProgram()** saapuminen sanomajonoon aiheuttaa funktion **quit()** suorittamisen, joka taas tarkoittaa sanomajonosta poistumisen ja ohjelman käytön lopettamisen.

- Signaalit kiinnitetään Qt:ssa **connect** funktion avulla esimerkiksi

  QObject::connect(objectMyClass, SIGNAL(finisProgram()), &a, SLOT(quit()), Qt::QueuedConnection);

- Lisää **connect** funktiosta ja kirjoitusasusta alla olevan linkin kautta
  http://doc.qt.io/qt-5/qobject.html#connect

- **connect** funktion parametrejä käsitellään pareittain siten, että ensimmäinen ja toinen parametri kuuluvat yhteen, ja parametrit kolmas ja neljäs kuuluvat yhteen.

- Esimerkissä siis luokasta **MyClass** täytyy löytyä signaali **finishProgram()** ja sehän on sinne itse tehty. Lisäksi luokasta **QCoreApplication** löytyy funktio **quit()** valmiina.

- Eli kun signaali **finishProgram()** suoritetaan, lisätään sanomajonoon funktion **quit()** suorittaminen, joka lopettaa sovelluksen.

- **Qt::QueuedConnection** parametrista kerrotaan sivulla
  http://doc.qt.io/qt-5/qt.html#ConnectionType-enum

- Suorita build ja aja ohjelmaa. Nyt ohjelman/prosessin pitäisi päättyä normaalisti.

```
#include <QCoreApplication>
#include "myclass.h"

int main(int argc, char *argv[])
{
   QCoreApplication a(argc, argv);

   MyClass *objectMyClass = new MyClass;

   QObject::connect(objectMyClass, SIGNAL(finishProgram()),
   &a, SLOT(quit()), Qt::QueuedConnection);

   int execReturnValue;

   objectMyClass->raiseFinishProgramSignal();

   execReturnValue = a.exec();

   cout << "Sanomajonosta poistuttu, koska sinne saapui viesti quit()" << endl;

   delete objectMyClass;
   objectMyClass = nullptr;

   return execReturnValue;
}
```

Olio-ohjelmointi: Qt –luokkakirjaston käyttö

- Lisää luokkaan signaali **void myFirstSignal();**

- Lisää luokan **public slot** osaan jäsenfunktio **void myFirstSlotFunction(); // tama on oma SLOT -jasenfunktio**

- Lisää yllä luodun **SLOT** -jäsenfunktion toteutukseen koodirivi
  cout << "Ollaan SLOT -jasenfunktiossa myFirstSlotFunction() koska signaali myFirstSignal() suoritettiin"<< endl;

- Yleensä **connect** funktiota käytetään luokan jäsenfunktioissa, koska kaikki **SIGNAL/SLOT** –rakenteet löytyvät luokista itsestään tai sitten erilaisten luokkien välisten yhteyksien kautta. Lisää luokan **MyClass** muodostinfunktioon alla olevat rivit:

```
// this osoitin osoittaa main() funktiossa luotuun olioon objectMyClass
connect(this, SIGNAL(myFirstSignal()), this, SLOT(myFirstSlotFunction()), Qt::QueuedConnection);

cout << "Signaali myFirstSignal() ylos"<< endl;
emit myFirstSignal(); // emit kaskylla nostetaan signaali ylos ja suoritetaan siihen liittyva SLOT-jasenfunktio.
```

- Eli nyt kun ohjelmassa luodaan olio luokasta **MyClass**, niin sen muodostinfunktiossa rakennetaan SIGNAL/SLOT yhteydet ja samalla suoritetaan signaalin ”nostaminen ylös”.

- Suorita build ja aja ohjelmaa.

Olio-ohjelmointi: Qt –luokkakirjaston käyttö, QDebug() funktio

- Kirjoita hakukenttään sana **QDebug**
- Avaa hakulistalta linkki (**QDebug Class | Qt Core …**)
- Mitä otsikkotiedosto täytyy lisätä ohjelmaan ja mikä paketti täytyy olla mukana **.pro** tiedostossa?
- Klikkaa ensimmäisessä kappaleessa linkkiä **more** ja selvitä luokan tarkoitus ja peruskäyttö (Basic Use)?
- Korvaa ohjelmassa kaikki **cout** käskyt **qDebug()** käskyllä ohjeen mukaisesti. Huomaa seuraavaa:
  - Luokan rakenteessa ei tarvita enää rivejä **#include <iostream>** ja **using namespace std;**
  - Tulostuskäskyssä ei tarvitse käyttää rivinvaihtoa **<< endl;** koodirivin lopussa.
- Suorita build ja aja ohjelmaa.

Olio-ohjelmointi: Qt –luokkakirjaston käyttö, QString

- Siirry takaisin selaimeen ja kirjoita hakukenttään sana **QString**
- Avaa hakulistalta linkki (**QString Class | Qt Core …**)
- Mitä otsikkotiedosto täytyy lisätä ohjelmaan ja mikä paketti täytyy olla mukana **.pro** tiedostossa?
- Klikkaa ensimmäisessä kappaleessa linkkiä **more** ja selvitä luokan tarkoitus?
- Siirry kohtaan **Initializing a String**

  - Huomaa, että esimerkin koodirivi **QString str = "Hello";** luo luokasta **QString** automaattisen olion nimeltä **str** ja antaa sille alkuarvon **Hello.**

Olio-ohjelmointi: Qt –luokkakirjaston käyttö

- Avaa **main()** funktio, ja lisää rivi **QString str = "Hello";** ennen riviä **int execReturnValue;** ja tulosta olion arvo **qDebug()** funktiolla.

- Lisää ohjelmaan rivi **int myValue = 15;** rivin **QString str = "Hello";** jälkeen.

- Etsi luokan **QString** dokumentaatiosta kohdasta **Public Functions** luokan jäsenfunktio, jolla voit asettaa edellä luodun **int** tyyppisen muuttujan arvon ohjelmassa olevaan **str** olion arvoksi ja tulostaa **str** olion arvon.