Лекция 2. Статические и динамические библиотеки. Немного про статические поля.

Автор: Набиев Марат

Библиотека — сборник подпрограмм или объектов, как правило, ориентированных на решение набора близких по тематике задач. С точки зрения их использования, они могут быть статическими и динамическими.

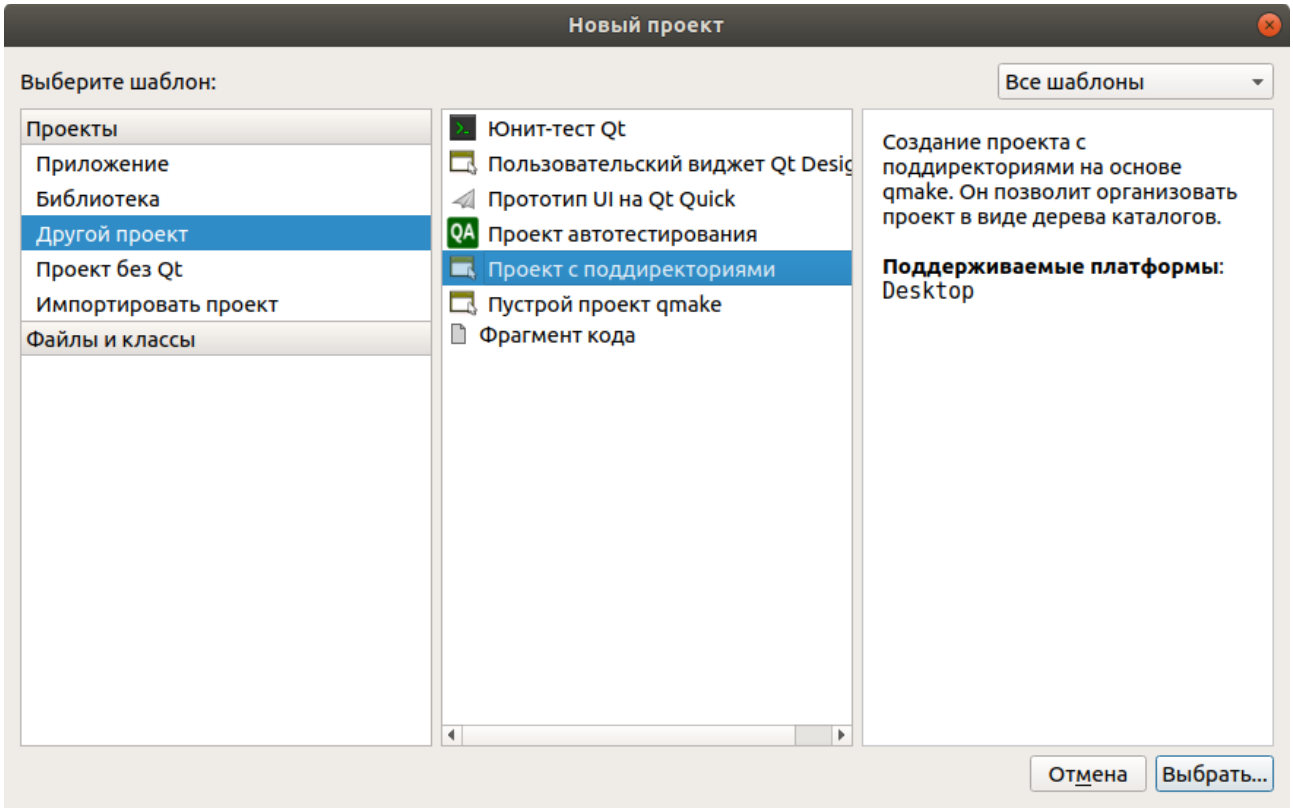
Статические библиотеки (static library) могут представлять собой набор исходных кодов, подключаемых программистом или заранее скомпилированные объектные файлы, связываемых вместе на этапе компиляции. В винде имеют расширение .lib. В результате подключения статической библиотеки, программа включает все используемые ею функции, из-за чего .exe становится больше, но программа более автономна.

Динамические библиотеки (shared library, dynamic link library) загружаются операционной системой по требованию запущенной программы, в ходе ее выполнения. Эта библиотека загружается один раз в память, и повторная загрузка уже не выполняется. Причем одна библиотека может использоваться несколькими программами.

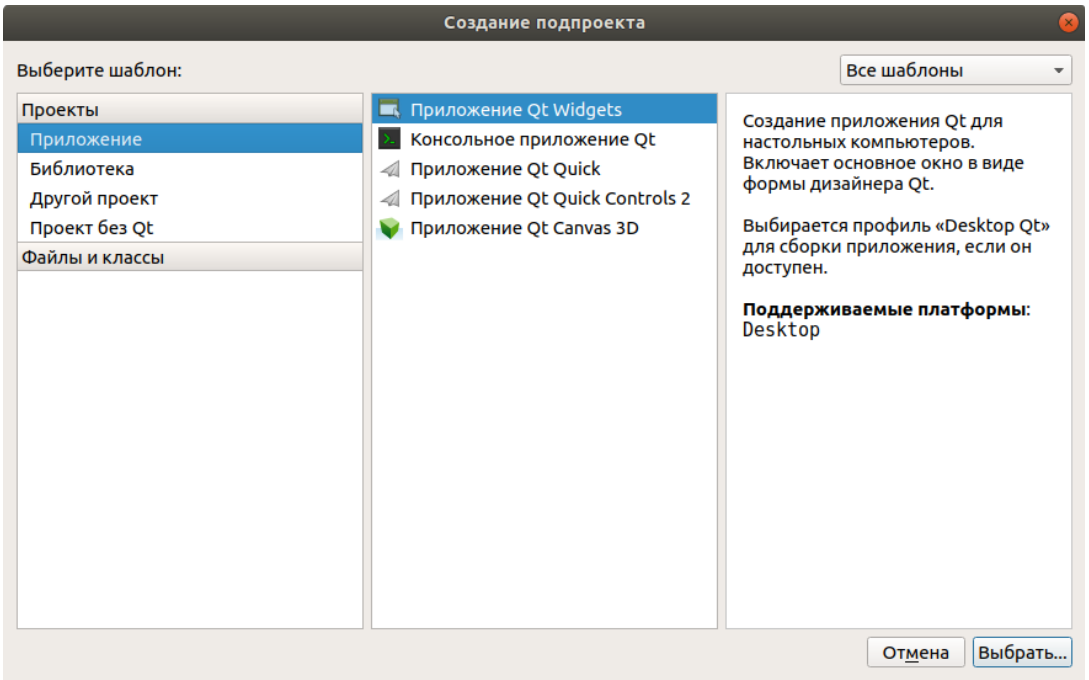
Пример создания статической и динамической библиотеки.
Создадим следующее простое приложение: будет окно, в котором будет текстовое поле, кнопка и 2 метки. В текстовое поле будем писать число, и при нажатии на кнопку в одной метке появится удвоенное число, а в другом утроенное. Удваивание и утраивание числа реализуем через отдельные библиотеки, одна будет статической, другая динамической (смысла в этом конечно нет, но ради тренировки пойдет)

1. Создадим в Qt проект с поддиректориями.

Другой проект → проект с поддиректориями → Выбрать

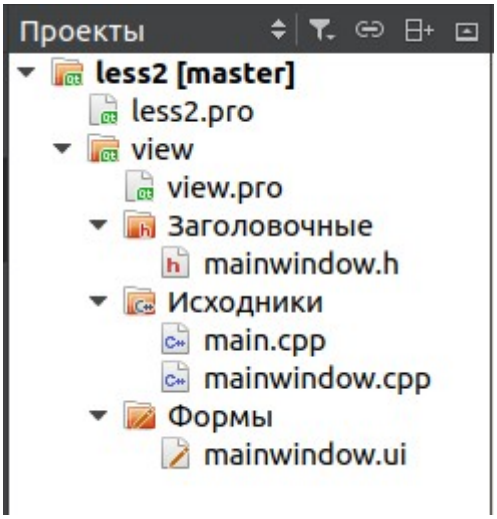


Там дальше пишем название, далее, далее, готово. Потом сразу нам будет предложено добавить подпроект. Здесь выбираем обычное Qt приложение как из прошлого занятия.



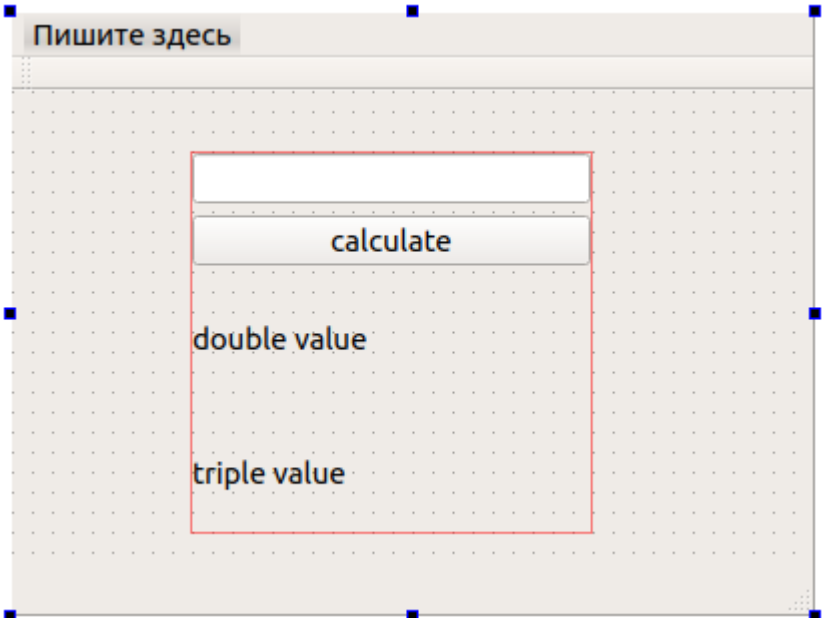
Пишем название, далее, далее готово.

И получили проект с подпроектом:



Далее рисуем по-быстрому форму. Для текстовое поле — `Line Edit`, кнопка `Push Button`, метки — `Label`. Эти элементы можете переименовать как вам удобно.

Рисуем примерно следующую форму:



Не буду акцентировать на этом внимание, т. к. в прошлом занятии довольно подробно разбиралось.

Далее пишем слот, для обработки нажатия кнопки:

```
 7  class MainWindow;
 8  }
 9
10  class MainWindow : public QMainWindow
11  {
12      Q_OBJECT
13
14  public:
15      explicit MainWindow(QWidget *parent = 0);
16      ~MainWindow();
17
18  private:
19      Ui::MainWindow *ui;
20  private slots:
21      void calculate();
22  };
```

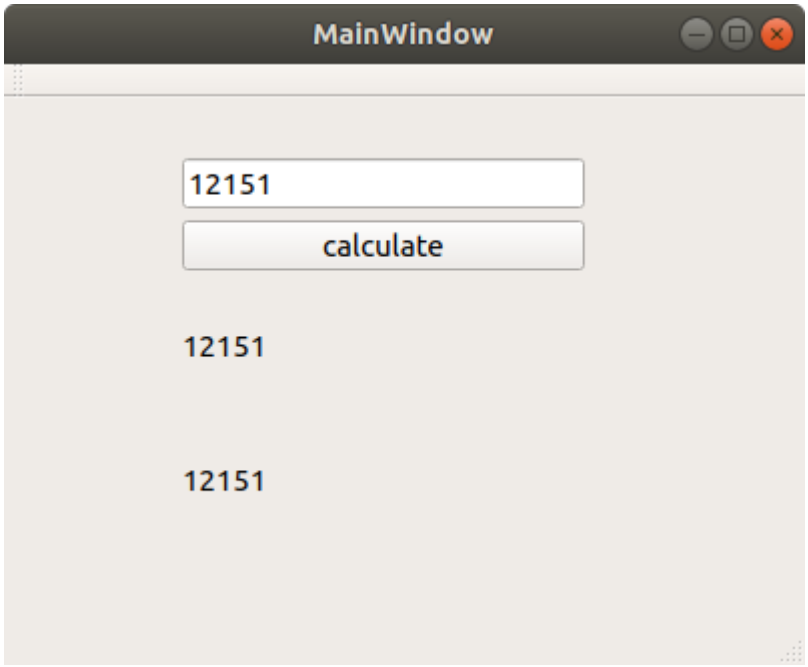
В ней будем получать значение из текстового поля, обрабатывать его и записывать метки.

Реализуем!

```
17  void MainWindow::calculate()
18  {
19      //считываем текст из поля и сразу переводим его в вещественное
20      double k = ui->lineEdit->text().toDouble();
21      double doubleK = k; //пока оставим так
22      double tripleK = k; //не бойтесь, скоро удвоим и утроим
23
24      //запишем в метки (я поменял названия меток)
25      //QString::number переводит из числа в строку
26      //это надо, т.к. setText принимает только число
27      ui->doubleLabel->setText(QString::number(doubleK));
28      ui->tripleLabel->setText(QString::number(tripleK));
29  }
```

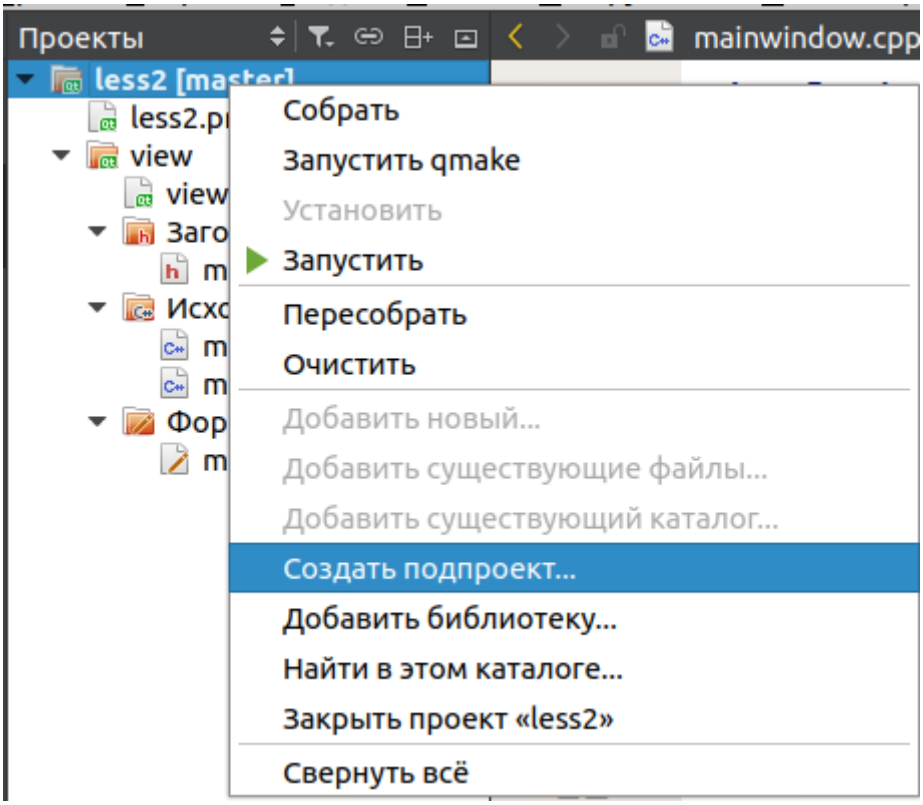
Не забудьте в конструкторе добавить `connect` сигнала кнопки с этим слотом :)

И в итоге будет такая штука:

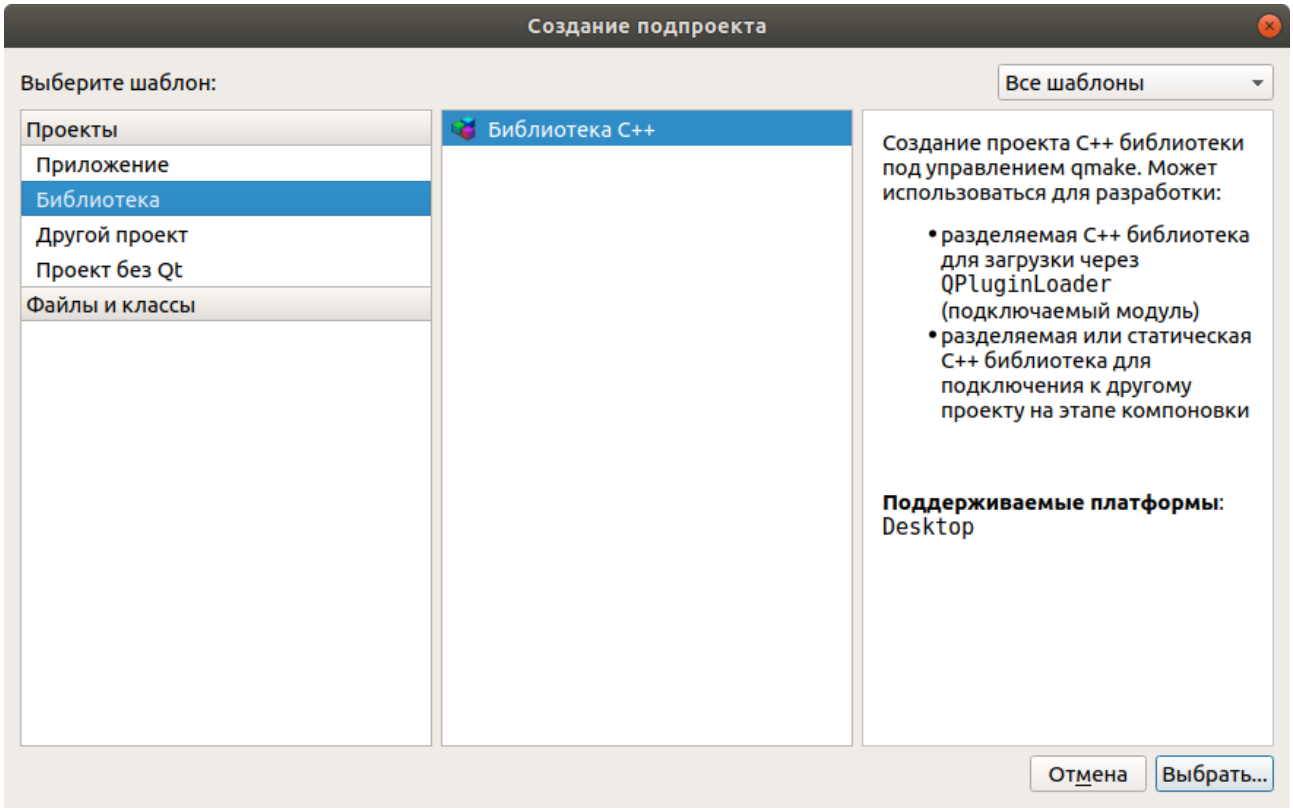


Но мы же хотели удваивать и утраивать число. Теперь реализуем это!

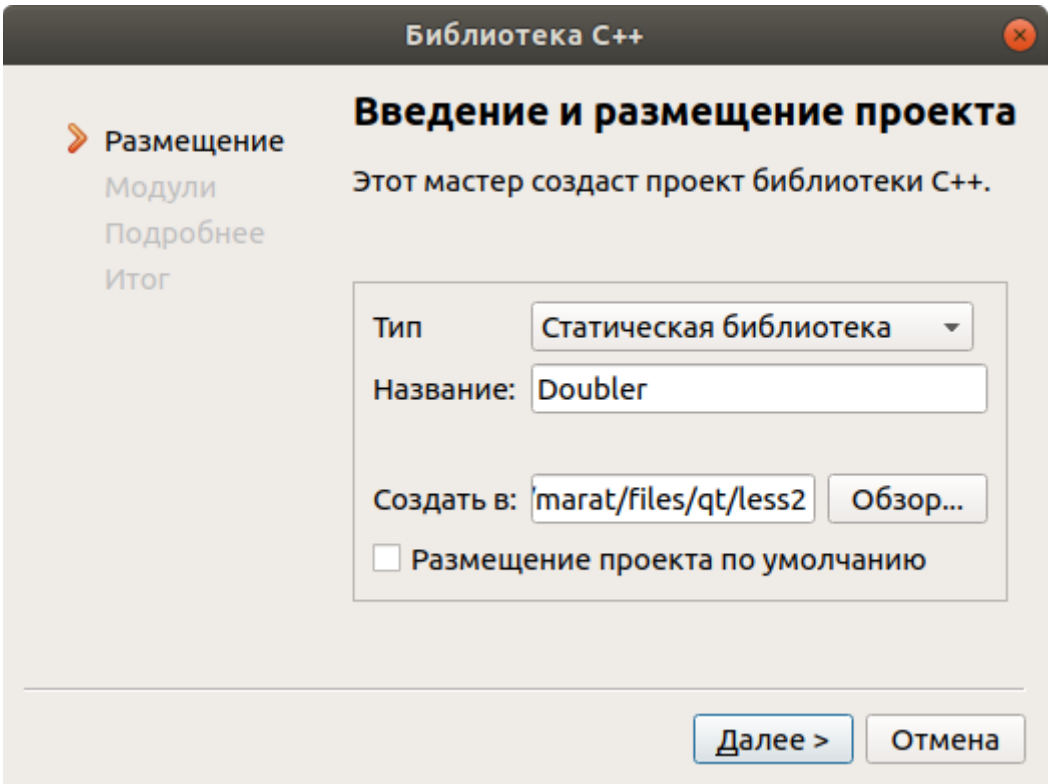
Нажимаете ПКМ на основной проект и там выбираете «создать подпроект»



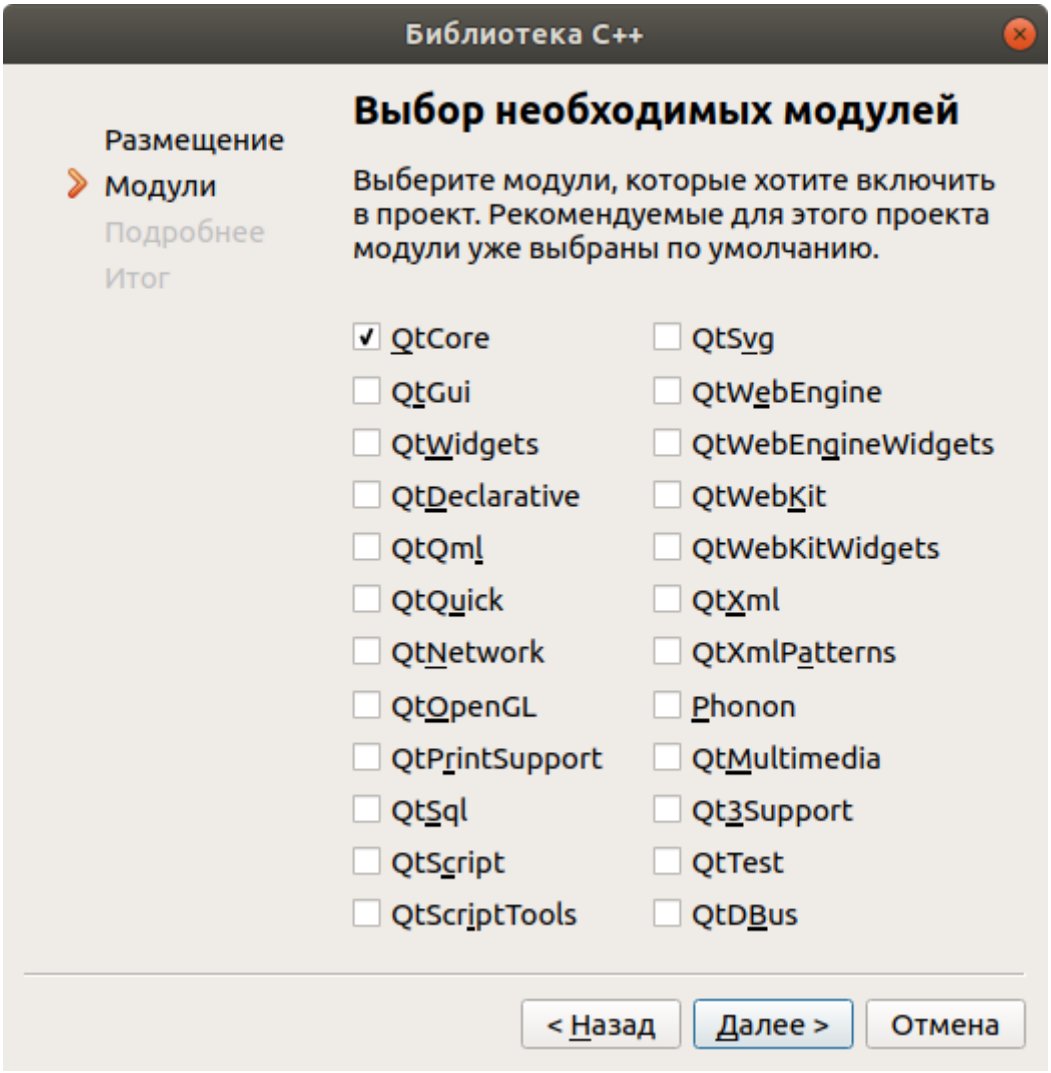
Там выбираете Библиотека → Библиотека C++



Далее выбираете тип «Статическая библиотека» и пишете название библиотеки



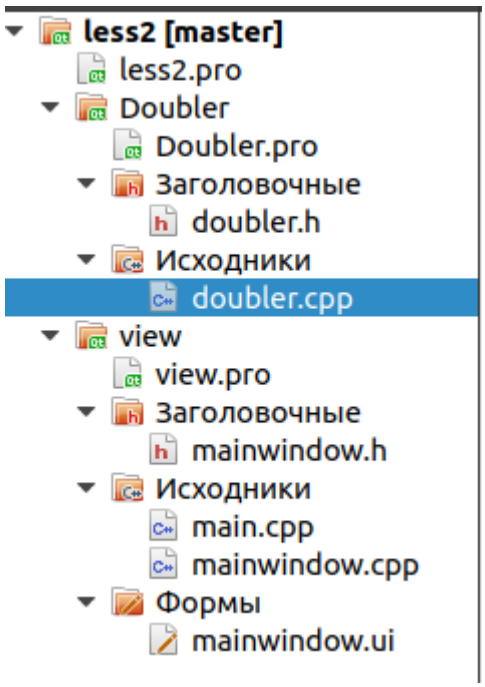
Нажимаете «Далее»



Нажимаете далее, здесь пока ничего не надо менять.

(Внимание, спойлер!!! В будущем, когда будете реализовывать View и Model, здесь нужно будет поставить пару галочек)
И далее, далее, готово.

Получаете еще один подпроект



У нас там есть один класс, реализуем в нем 1 статическую функцию

Лирическое отступление.

Что такое статические поля класса.

Некоторые функции могут принадлежать не конкретному объекту, а всему классу. Такие члены класса разделяются между всему экземплярами данного класса и хранятся в одном месте, а для вызова методов класса не надо создавать объект.

Пример будто из жизни.

Допустим у нас есть класс Автомобиль ВАЗ 2101 или в простонародье - Копейка. Чертеж машины будет для всех копеек одинаковый, это статическое поле, но у каждой машины могут быть свои отличительные черты, например, цвет, номер кузова и т. д.. Т.к. эти свойства для каждой машины индивидуальны, они не статические.

Продолжим.

Добавим статический метод. Просто перед методом надо написать ключевое слово `static`

```
5  class Doubler
6  {
7
8  public:
9      Doubler();
10     static double calc(double k);
11 };
```

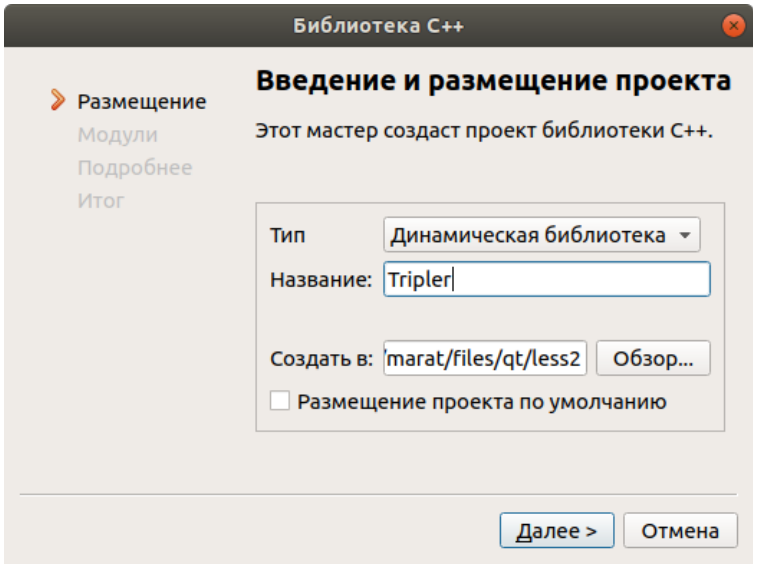
Реализуем этот метод

```
8  double Doubler::calc(double k)
9  {
10     return 2*k;
11 }
```

Теперь создадим динамическую библиотеку, в которой будет утраивать число.

Для этого также нажимаем правой кнопкой по основному проекту, нажимаем на «Создать подпроект», выбираем «Библиотека» → «Библиотека C++».

Но здесь уже тип выбираем «Динамическая библитека»



И там далее, далее, готово.

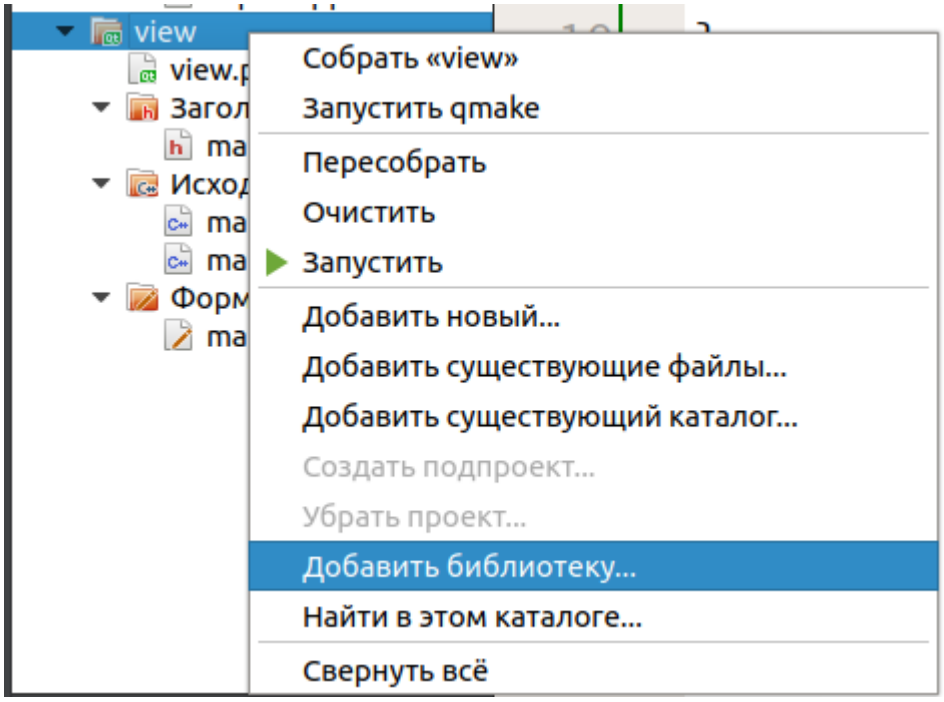
Так же добавим статический метод и реализуем его

```
6  class TRIPLERSHARED_EXPORT Tripler
7  {
8
9  public:
10     Tripler();
11     static double calc(double k);
12 };
```

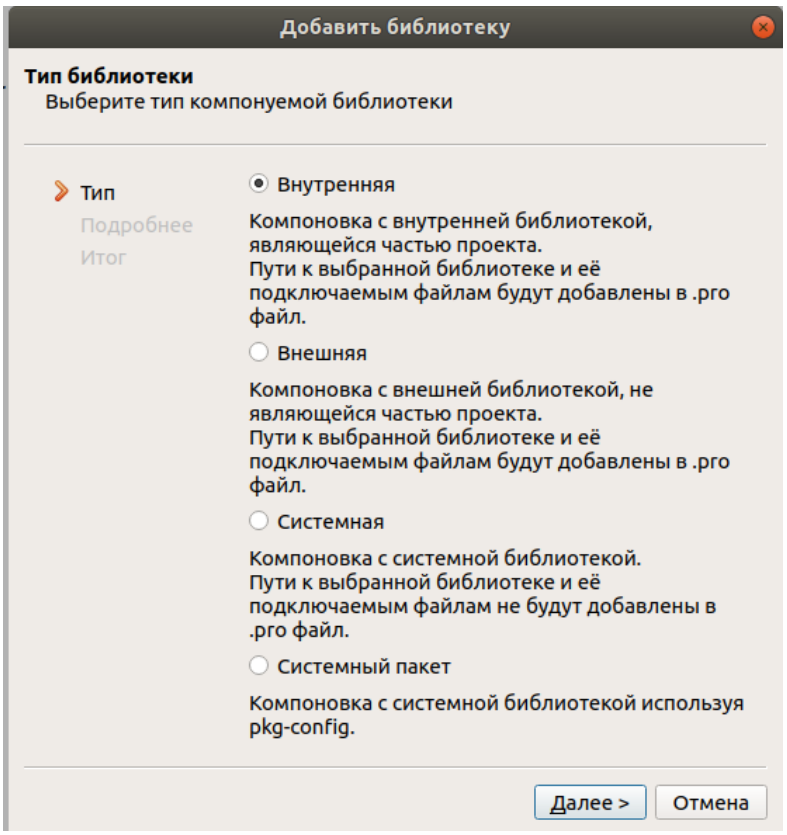
```
7  double Tripler::calc(double k)
8  {
9      return 3*k;
10 }
```

Не пугайтесь страшных слов перед названием класса, это нужно для экпорта класса из нашей динамической библиотеки.

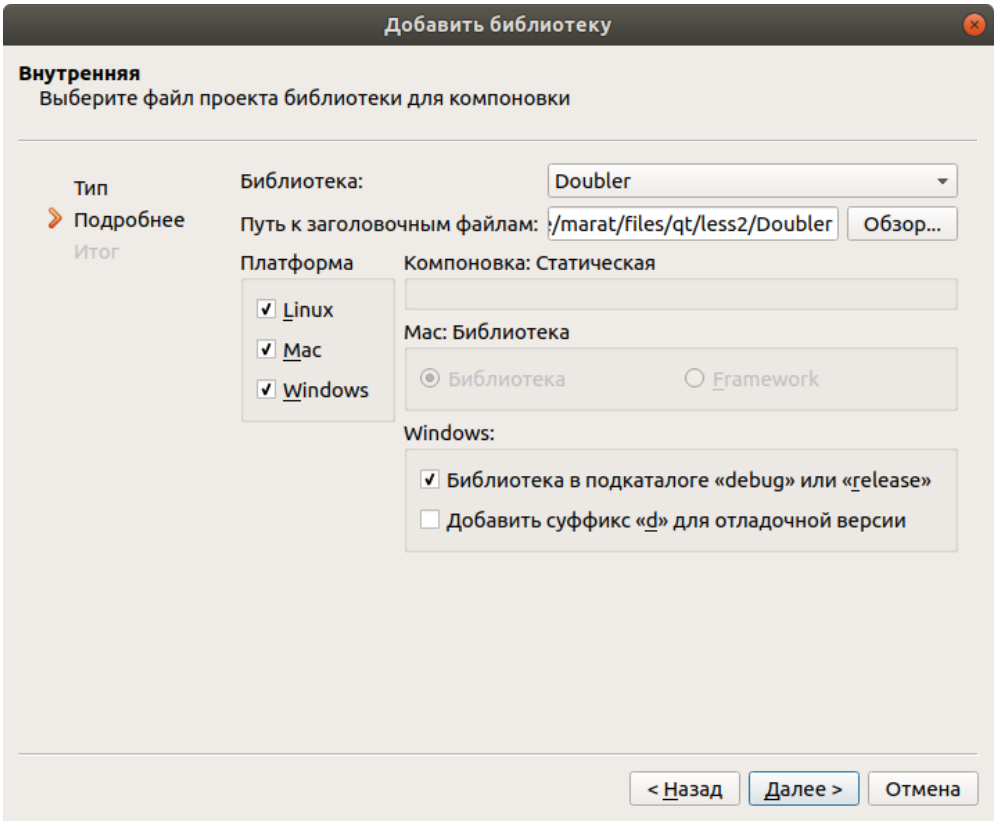
Теперь сделаем так, чтобы наш подпроект, в котором мы хотели удваивать и утраивать «видел» эти библиотеки. Для этого нажимаем ПКМ на наш подпроект и выбираем «добавить библиотеку»

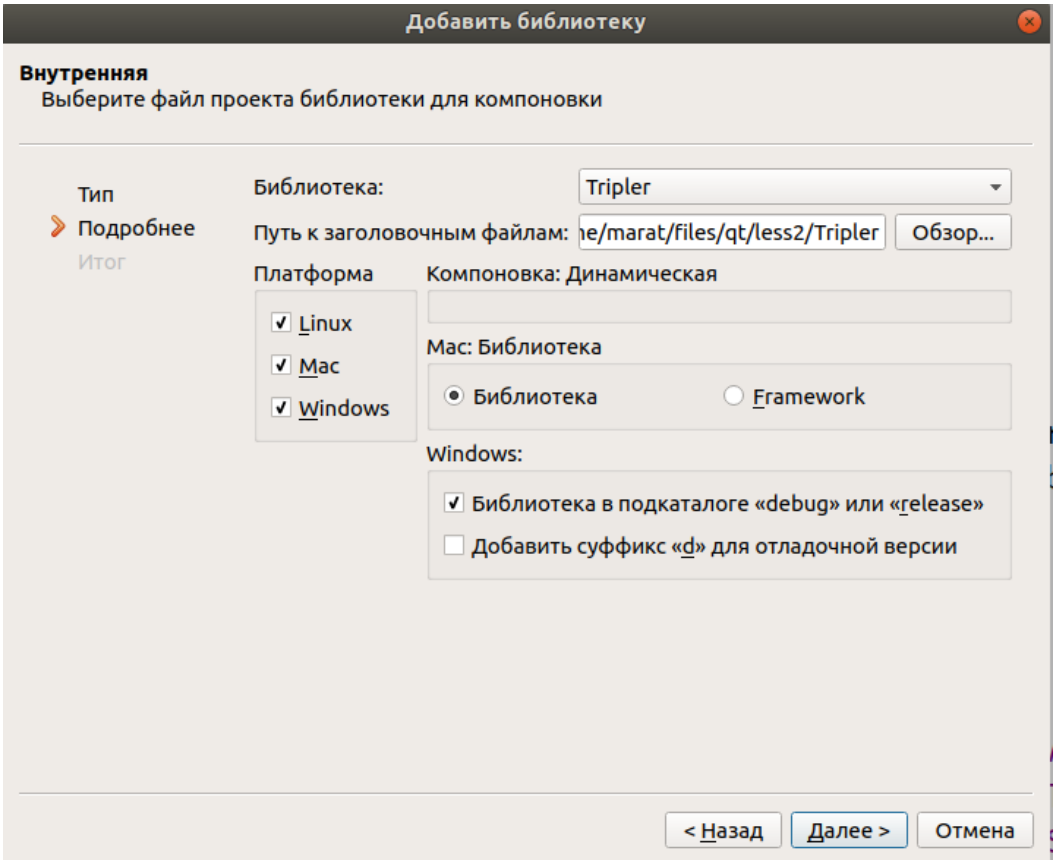


Далее выбираем «Внутренняя»



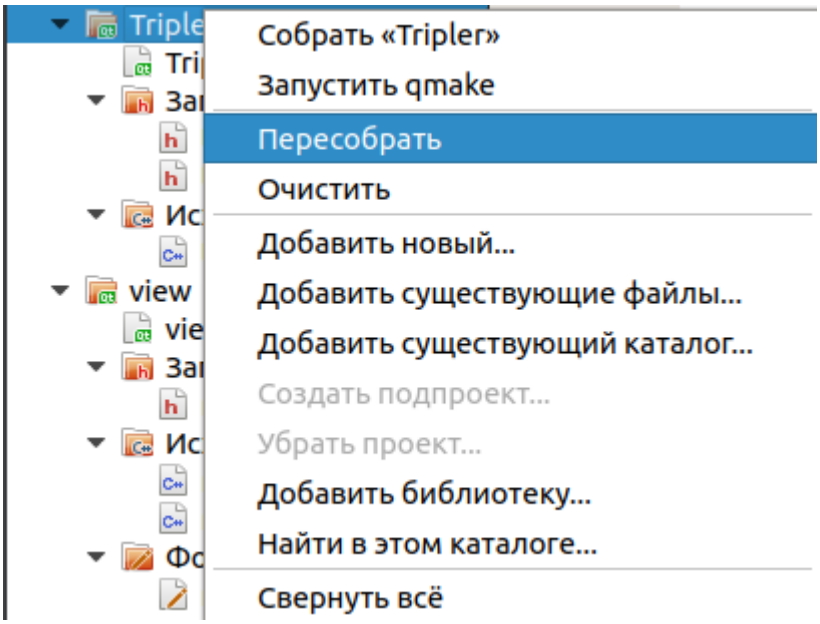
И там добавляем сначала Doubler, а потом Tripler





И наш .pro файл автоматически добавит нужные настройки. Но если вы не доверяете, можете все прописывать сами.

Теперь надо пересобрать наши библиотеки, для этого нажимаем по библиотеке ПКМ и выбираем пересобрать.



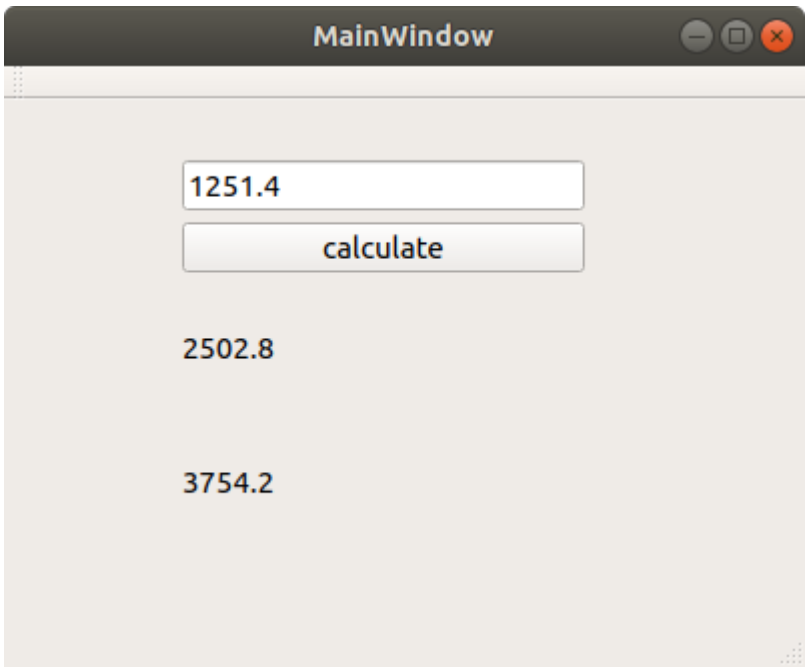
Если мы перейдем в `mainwindow.cpp`, то там сможем написать `#include "doubler.h"` и `#include "tripler.h"`

```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3  #include "doubler.h"
4  #include "tripler.h"
```

И воспользоваться классами из этой библиотеки для своих нужд.

```
18  void MainWindow::calculate()
19  {
20      //считываем текст из поля и сразу переводим его в вещественное
21      double k = ui->lineEdit->text().toDouble();
22      //вызываем статические методы наших классов
23      double doubleK = Doubler::calc(k);
24      double tripleK = Tripler::calc(k);
25
26      //запишем в метки (я поменял названия меток)
27      //QString::number переводит из числа в строку
28      //это надо, т.к. setText принимает только число
29      ui->doubleLabel->setText(QString::number(doubleK));
30      ui->tripleLabel->setText(QString::number(tripleK));
31  }
```

И если все правильно, то при нажатии на calculate у нас должно вычисляться то, что мы хотели.



Ссылка на исходники: https://github.com/nma2207/qt-lessions/tree/master/less2

Следующее занятие про MVP. Введение и попробуем создать простое приложение с его использованием.