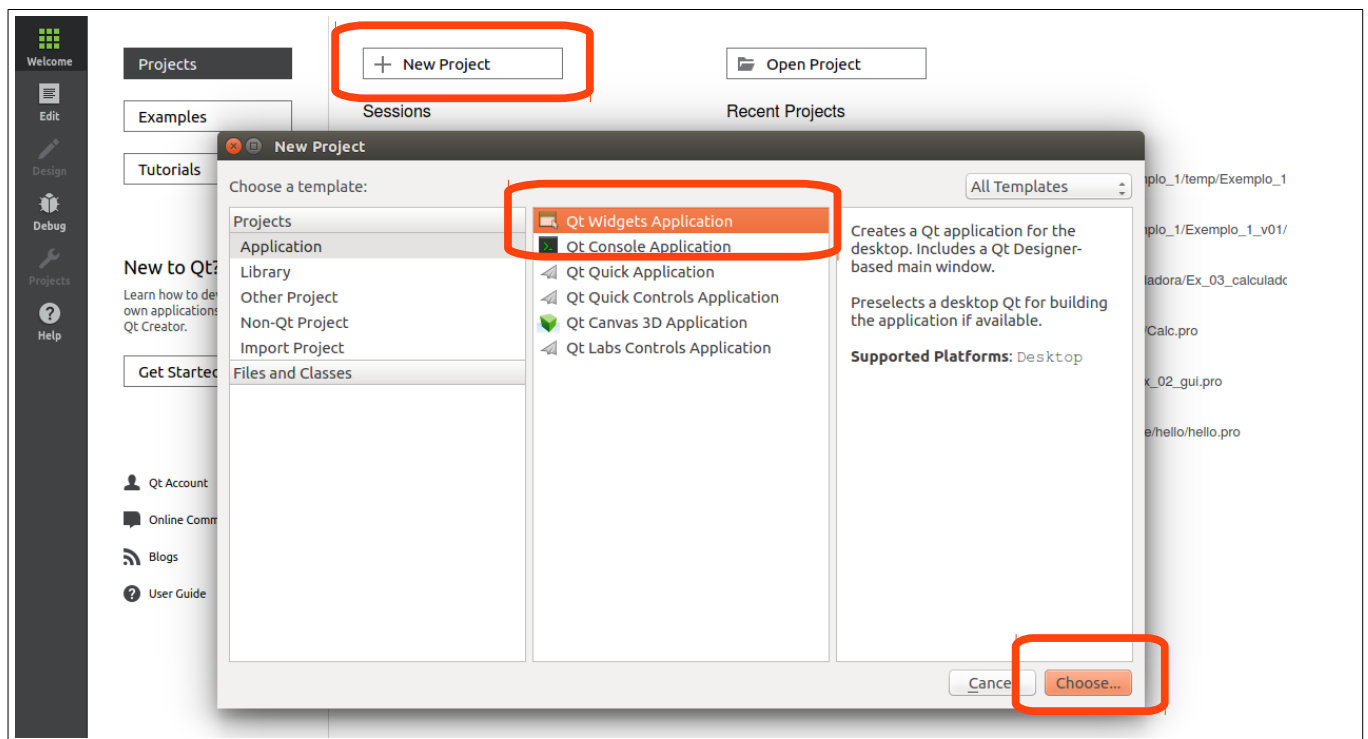


Exemplo 1: Notepad (editor de texto)

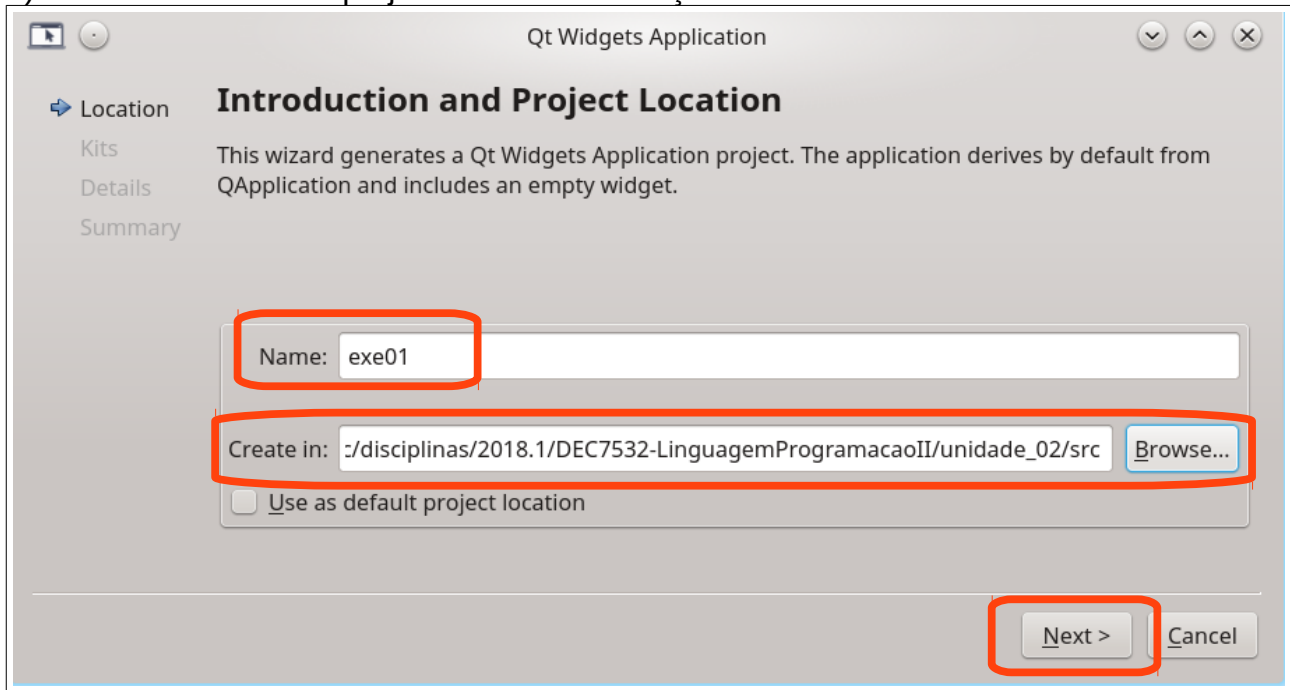
Nesse nosso primeiro exemplo vamos usar **Qt Creator IDE** e **Qt Designer** para gerar algumas partes do código, mas esse código também poderia ser escrito “manualmente”.

Primeiro vamos criar um novo projeto junto com todos os arquivos necessários. Depois vamos usar o **Qt Designer** para modificar a parte da interface gráfica com usuário (GUI).

- 1) Criar um novo projeto do tipo **QT Widgets Application** no **QT Creator**:
File > New File or Project > Applications > Qt Widgets Application > Choose

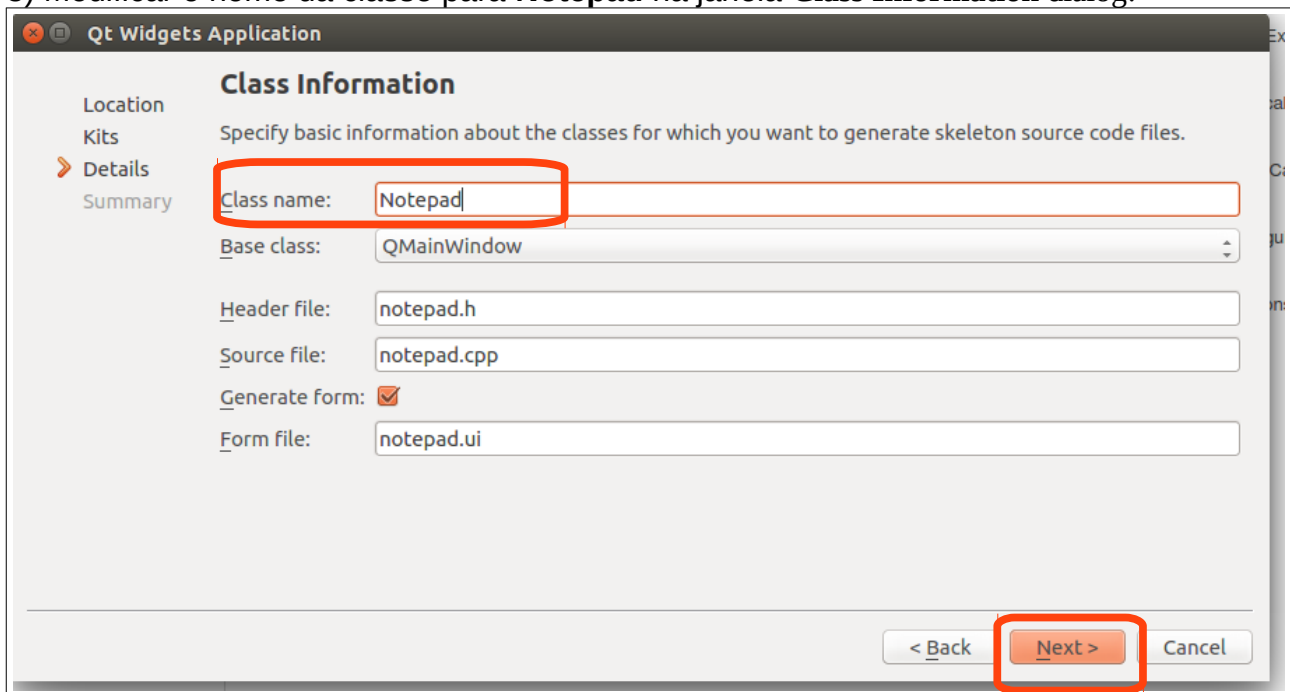


2) Escolher o Nome do projeto e a sua localização no HD:



Seguir para os próximos 2 passos.

3) Modificar o nome da classe para **Notepad** na janela **Class Information** dialog:



O Wizard vai criar um projeto composto por arquivo principal (main) e um conjunto de arquivos responsáveis por interface gráfica (Notepad widget):

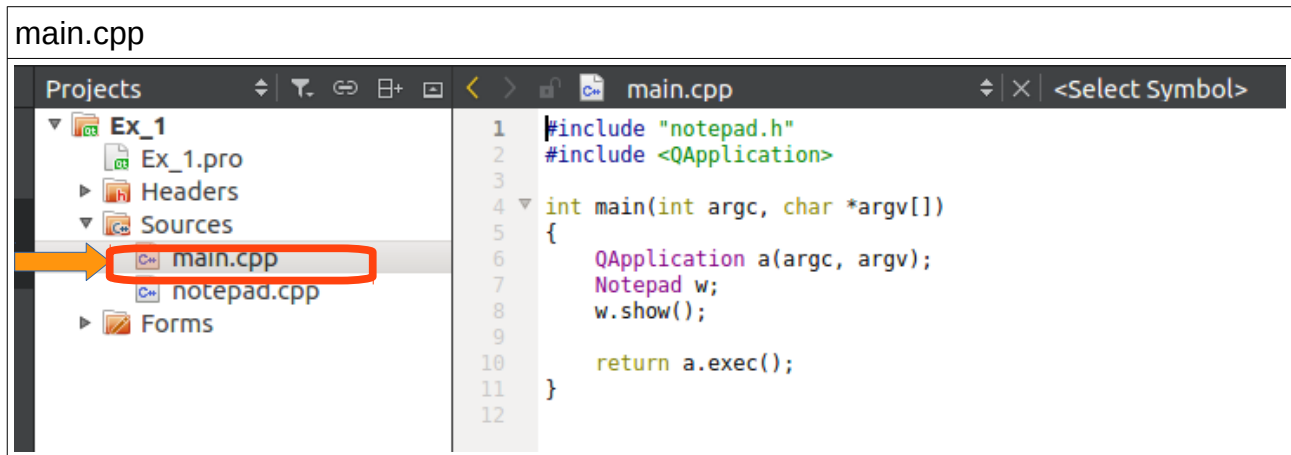
- notepad.pro – o arquivo do projeto
- main.cpp – arquivo-fonte (the main source file) do aplicativo
- notepad.cpp - arquivo-fonte (the source file) para classe **notepad** do **Notepad**

widget.

- notepad.h – arquivo-cabeçalho (the header file) correspondente para classe **notepad** do **Notepad widget**.
- notepad.ui – componente forma (the UI form) para **Notepad widget**

Main Source File

O arquivo main.cpp será gerado automaticamente e terá o seguinte conteúdo:



6 `QApplication a(argc, argv);` cria um objeto do tipo [QApplication](#). Esse objeto será responsável pela administração de recursos necessários para executar qualquer programa em **QT** que usa **widgets**.

7 `Notepad w;` cria um objeto do tipo Notepad.
Para esse objeto que o Wizard cria a classe e o arquivo UI.
A interface gráfica contém elementos visuais, chamados **widgets** em **QT**.
Um **widget** pode conter outros **widgets**.

Os widgets são invisíveis por definição.
Para tornar um widget visível é usada função **show()**

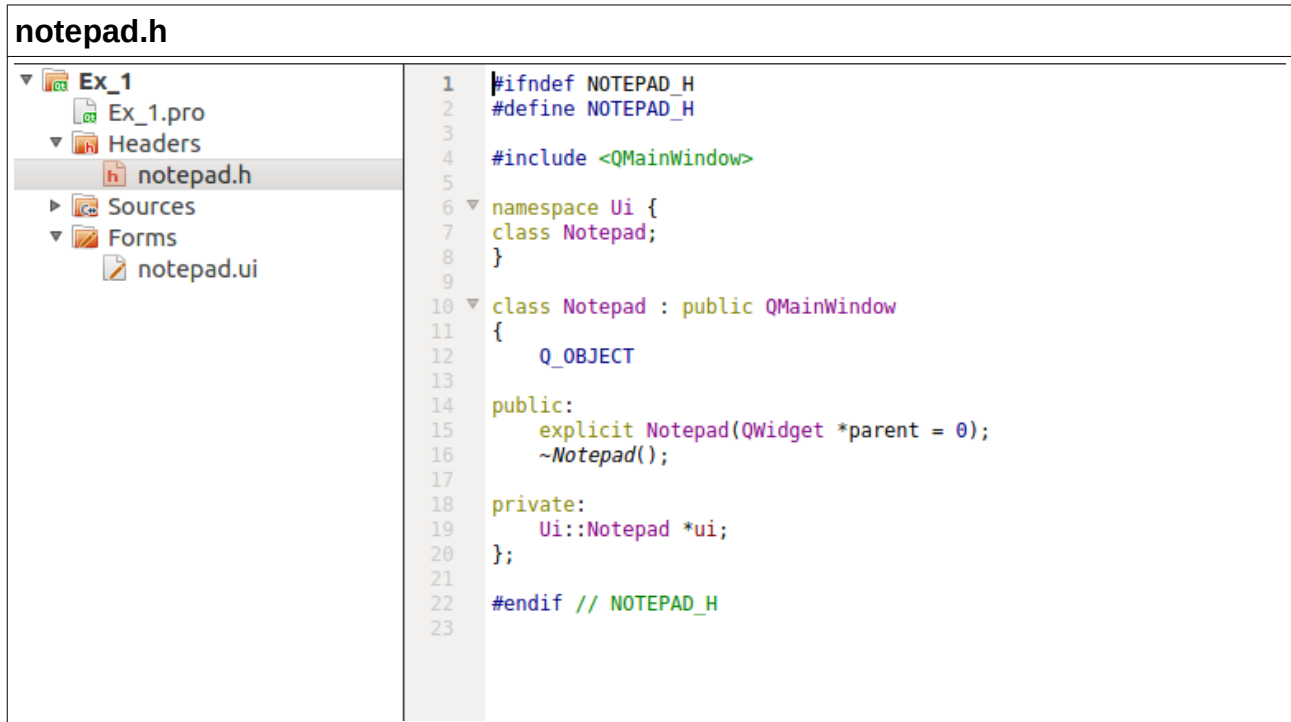
8 `w.show();`

10 `return a.exec();` faz com que aplicativo entra em modo de execução e análise dos eventos (como movimento do cursor do mouse ou aperto de teclas)

Notepad Header File

Wizard gera um header (arquivo-cabeçalho) correspondente para classe Notepad que necessariamente inclui:

- construtor
- destrutor
- objeto UI



4 `#include <QMainWindow>` inclui [QMainWindow](#) que providencia a janela principal do programa.

6 `namespace Ui {`
7 `class Notepad;`
8 `}` declara a classe **Notepad** em **Ui namespace**, que é o padrão para classes UI gerados a partir dos arquivos **.ui** por ferramenta **uic tool**

10 `class Notepad : public QMainWindow`
11 `{`
12 `Q_OBJECT` declaração da classe que contém o macro **Q_OBJECT**.

Essa declaração deve aparecer logo no início da declaração da classe, e declara a nossa classe como [QObject](#).

Naturalmente isso permite que a nossa classe se torna herdeira da [QObject](#).

[QObject](#) permite estender as classes estilo C++ acrescentando algumas coisas adicionais.

```
14 public:
15     explicit Notepad(QWidget *parent = 0);
```

declara um construtor que tem um argumento padrão chamado **parent**.

O valor **0** indica que nosso widget não tem **parent** (não tem pai), ou seja ele é um widget que esta no topo da hierarquia.

```
16     ~Notepad();
```

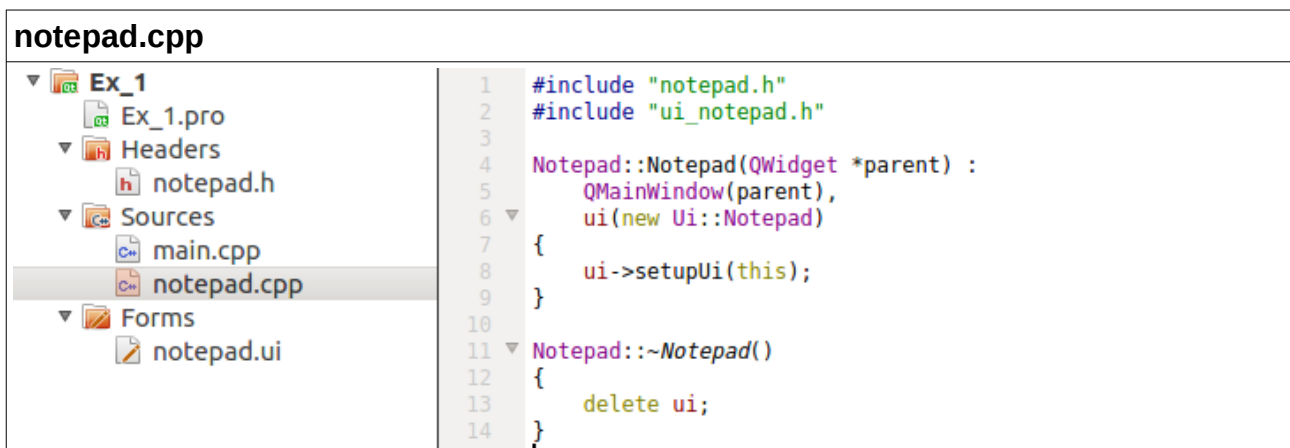
declara um destrutor virtual para liberar os recursos usados durante a execução do programa.

```
18 private:
19     Ui::Notepad *ui;
```

declara uma variável, que é ponteiro para classe **Notepad** UI.

Notepad Source File

O arquivo-fonte para classe Notepad também é gerado automaticamente pelo Wizard:



```
1 #include "notepad.h"
2 #include "ui_notepad.h"
```

inclui o **header** da classe **Notepad** que foi gerado pelo **Wizard** e o **UI header** que foi gerado pela ferramenta **uic tool**

```
4 Notepad::Notepad(QWidget *parent) :
```

define o construtor para classe **Notepad**

```
    QMainWindow(parent),
```

chama o construtor do [QMainWindow](#) , que é a classe base da classe **Notepad**

```
6     ui(new Ui::Notepad)
```

cria uma instância da **classe UI** e atribui ela para membro **ui**

```
8     ui->setupUi(this);
```

faz o setup do UI

```

11 Notepad::~Notepad()
12 {
13     delete ui;
14 }

```

destrutor que aplica a operação delete ao ui

Desenvolvimento de interface gráfica

Wizard gera definição da interface gráfica com usuário em formato XML em arquivo chamado **notepad.ui**.

Quando abrimos **notepad.ui** em **Qt Creator** ele automaticamente abre seu **Qt Designer** integrado.

Quando criamos nossa aplicação **Qt Creator** aciona Qt [User Interface Compiler \(uic\)](#) que lê arquivos do tipo **.ui** e cria código C++ correspondente.

Qt Designer

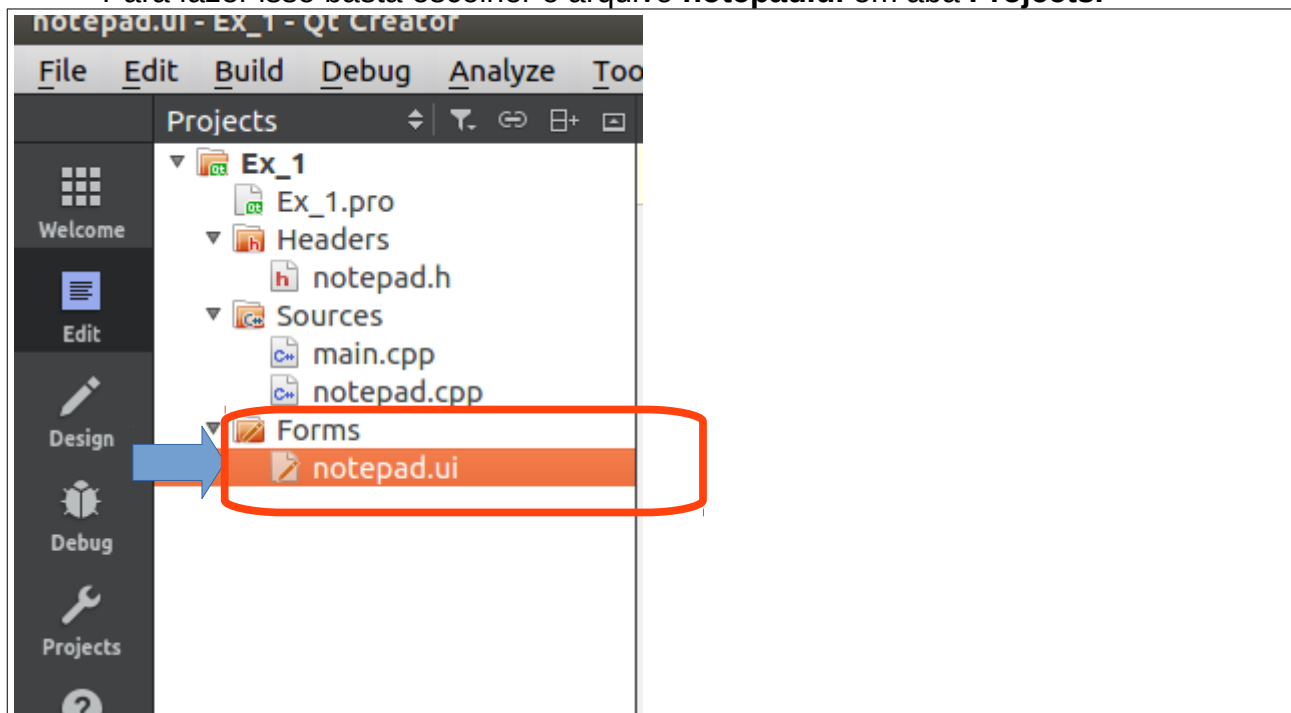
Wizard cria aplicativo que usa [QMainWindow](#), que tem seu próprio layout. Nele podemos adicionar nossos próprios botões, menus e outros componentes.

4) Vamos adicionar dois componentes na nossa janela principal:

- [QTextEdit](#)
widget (componente) de edição de texto. Quando digitamos texto nesse widget ele analisa botões de teclado pressionados (recebe essa informação como evento) e responde desenhando letras correspondentes na tela.
- [QPushButton](#)
widget do tipo “boton”, que vai permitir a saída do nosso aplicativo quando pressionado (ou seja acionado por um clique de mouse)

Para adicionar esse componentes precisamos entrar em modo **Qt Designer**.

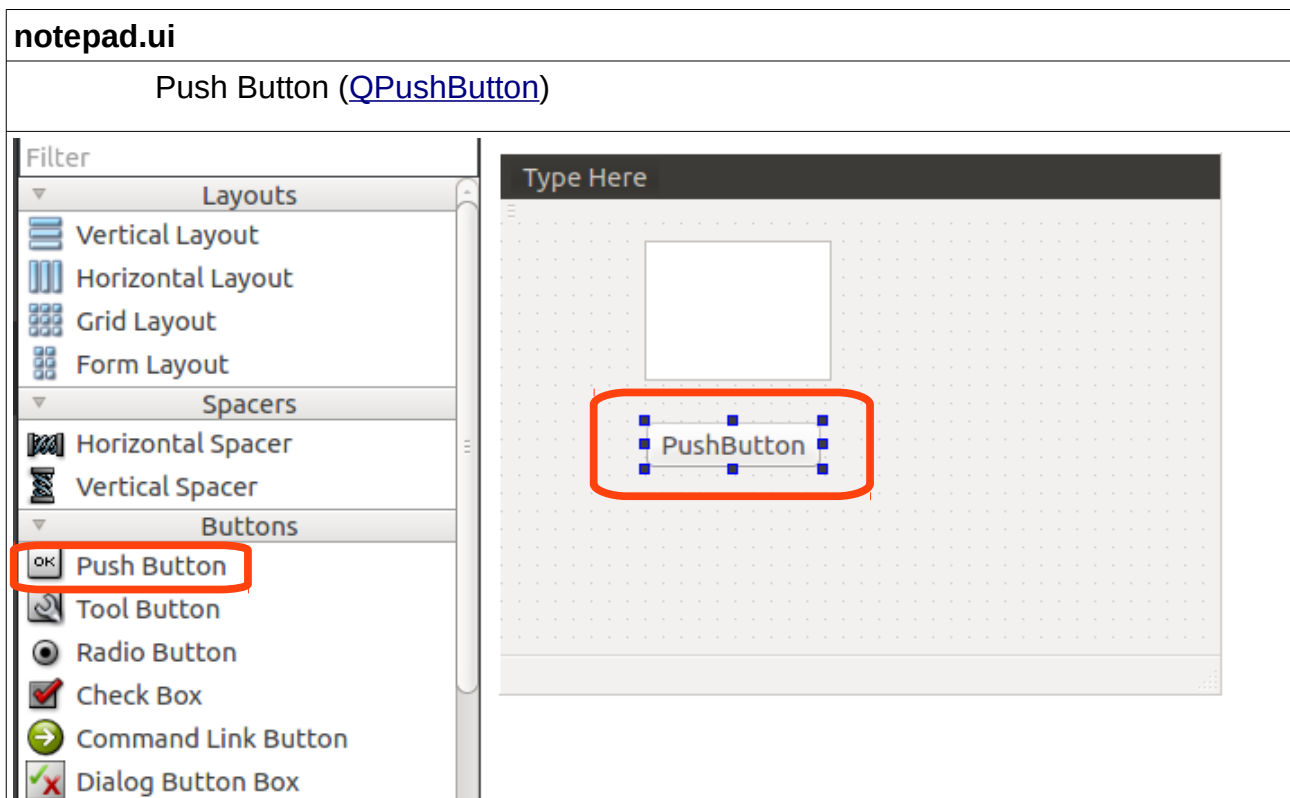
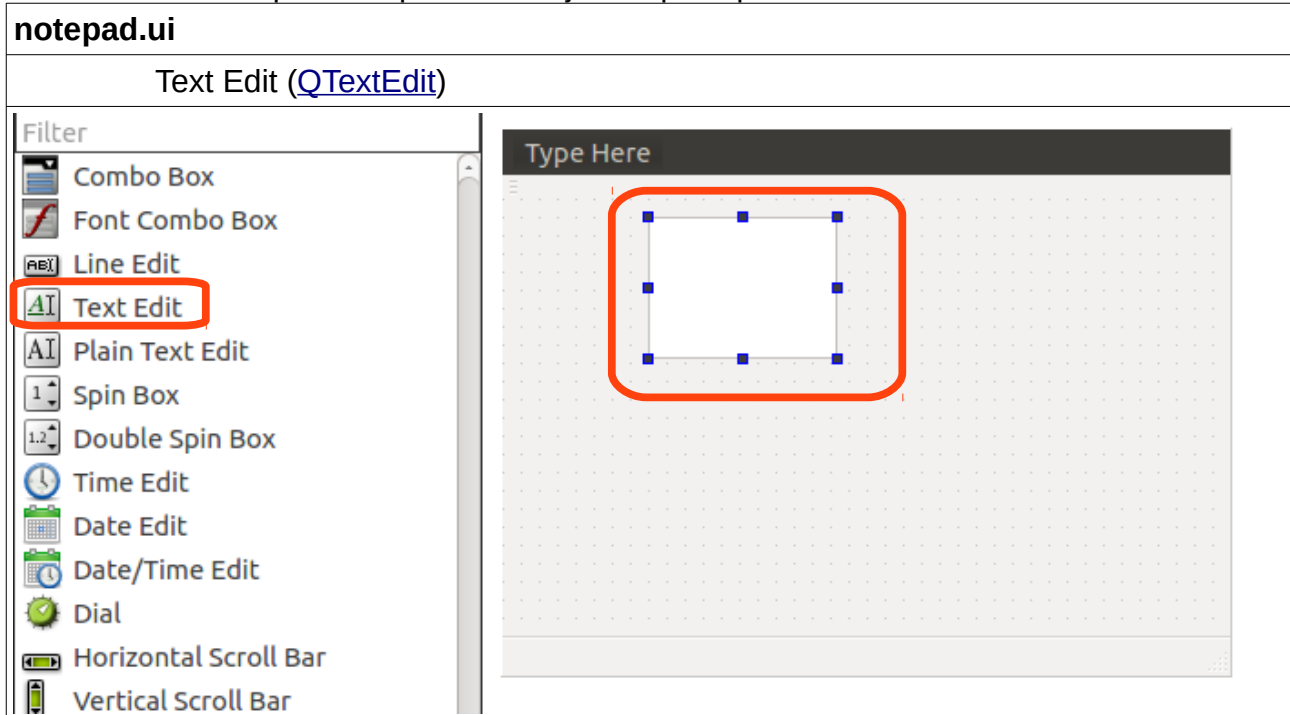
Para fazer isso basta escolher o arquivo **notepad.ui** em aba **Projects**.



Feito isso precisamos escolher componentes

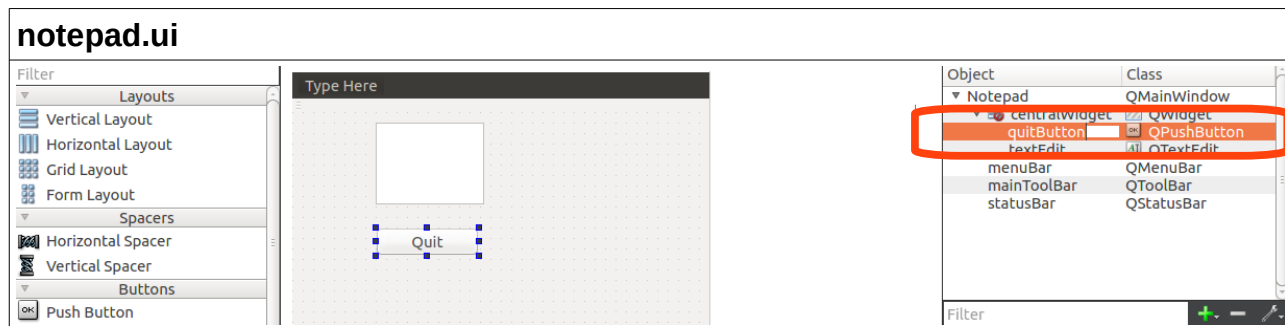
- Text Edit ([QTextEdit](#))
- Push Button ([QPushButton](#))

e arrastar eles um por uma para nossa janela principal

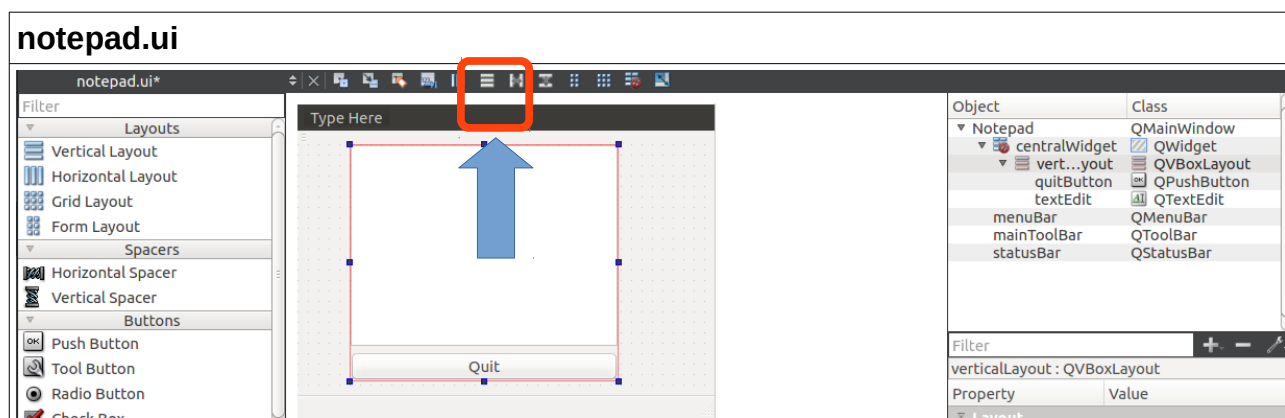


5) Clicar duas vezes no **Push Button** widget e digitar texto **Quit** (ou **Sair**).

6) No painel **Properties** trocar o valor do **objectName** para **quitButton**.



7) Apertar **Ctrl+A** (or **Cmd+A**) para selecionar todos os widgets e clicar em **Layout Vertically** (ou apertar **Ctrl+L**) para aplicar o layout vertical (posicionar com alinhamento vertical)



8) Apertar **Ctrl+S** (ou **Cmd+S**) para salvar as modificações

Project File

O arquivo do projeto *.pro é gerado automaticamente pelo Wizard e especifica os arquivos que fazem parte do projeto e algumas instruções para **qmake**.

Interação com usuário

O aplicativo que criamos ainda não executa nenhuma tarefa específica.

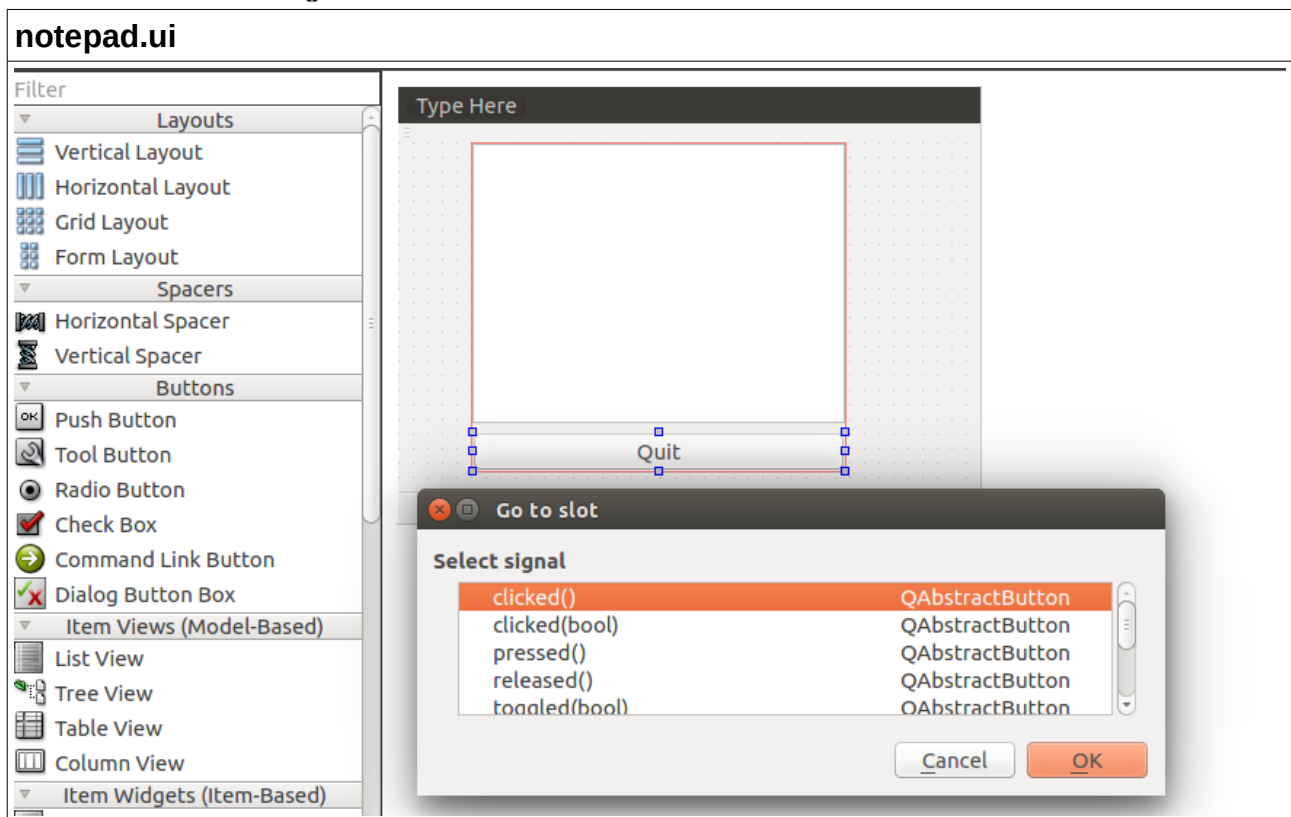
Vamos fazer com que botton que criamos, quando apertado faça fechar o nosso programa.

Para fazer isso vamos usar o mecanismo de **Signals and Slots**.

Um sinal (**Signal**) é emitido quando acontece um tipo específico de evento e **Slot** é a função que é chamada em resposta para esse sinal.

Existem alguns sinais e slots já predefinidos para **Qt widgets**. Podemos usar eles diretamente do **Qt Designer**.

9) Para usar o **Qt Designer** temos que chamar menu em cima do boton Quit e selecionar **Go to slot > clicked()**.



Um **private slot**, **on_quitButton_clicked()**, será adicionado para **header file** do **Notepad widget class (notepad.h)**.

notepad.h

▼ Ex_1

Ex_1.pro

▼ Headers

notepad.h

▼ Sources

main.cpp

notepad.cpp

▼ Forms

notepad.ui

```
1  #ifndef NOTEPAD_H
2  #define NOTEPAD_H
3
4  #include <QMainWindow>
5
6  namespace Ui {
7  class Notepad;
8  }
9
10 class Notepad : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     explicit Notepad(QWidget *parent = 0);
16     ~Notepad();
17
18     private slots:
19         void on_quitButton_clicked();
20
21 private:
22     Ui::Notepad *ui;
23 };
24
25 #endif // NOTEPAD_H
26
```

Uma função do tipo **private**, **Notepad::on_quitButton_clicked()**, será adicionada para **source file** da classe **Notepad (notepad.cpp)**.

notepad.cpp

▼ Ex_1

Ex_1.pro

▼ Headers

notepad.h

▼ Sources

main.cpp

notepad.cpp

▼ Forms

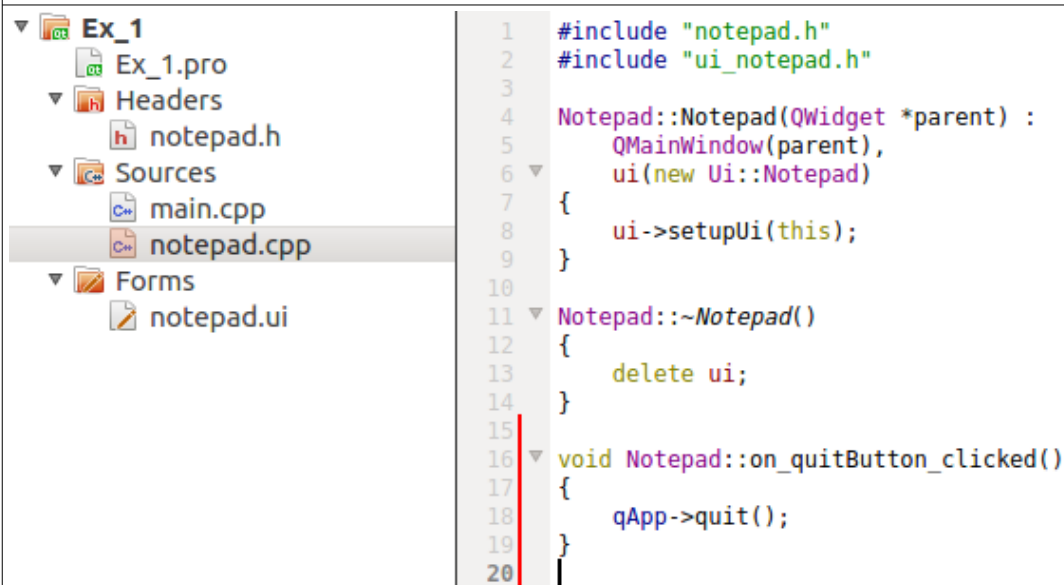
notepad.ui

```
1  #include "notepad.h"
2  #include "ui_notepad.h"
3
4  Notepad::Notepad(QWidget *parent) :
5      QMainWindow(parent),
6      ui(new Ui::Notepad)
7  {
8      ui->setupUi(this);
9  }
10
11 Notepad::~Notepad()
12 {
13     delete ui;
14 }
15
16 void Notepad::on_quitButton_clicked()
17 {
18     |
19 }
```

10) Agora precisamos escrever o código que fará com que o nosso programa feche e vamos fazer isso em nosso arquivo **source**:

```
qApp->quit();
```

notepad.cpp



The screenshot shows the Qt Creator interface. On the left, the Project Explorer displays the project structure: Ex_1 (containing Ex_1.pro), Headers (containing notepad.h), Sources (containing main.cpp and notepad.cpp), and Forms (containing notepad.ui). The notepad.cpp file is selected. The main editor shows the following C++ code:

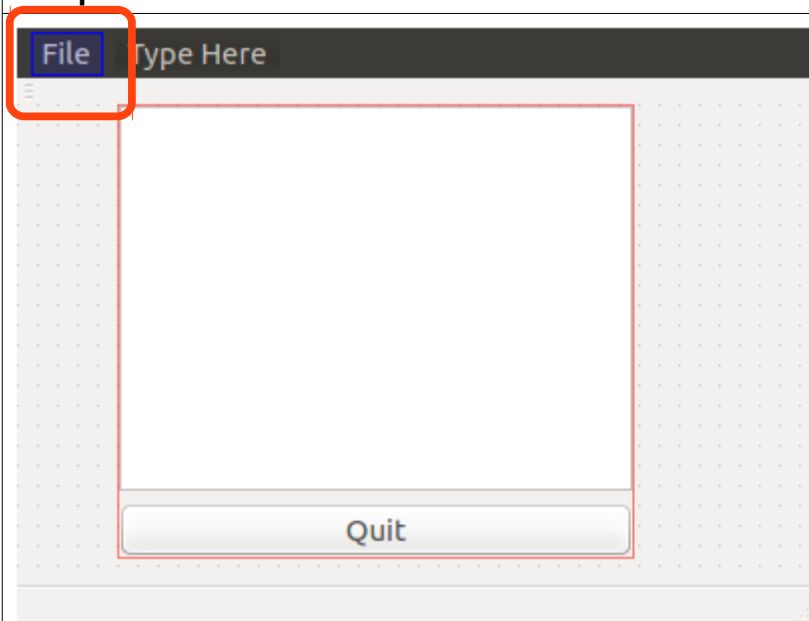
```
1  #include "notepad.h"
2  #include "ui_notepad.h"
3
4  Notepad::Notepad(QWidget *parent) :
5      QMainWindow(parent),
6      ui(new Ui::Notepad)
7  {
8      ui->setupUi(this);
9  }
10
11 ~Notepad()
12 {
13     delete ui;
14 }
15
16 void Notepad::on_quitButton_clicked()
17 {
18     qApp->quit();
19 }
20
```

11) Menu para Abrir e Salvar arquivos

Vamos usar **Qt Designer** para adicionar os widgets para interface.

No campo “**Type Here**” do menu padrão vamos digitar “**File**”

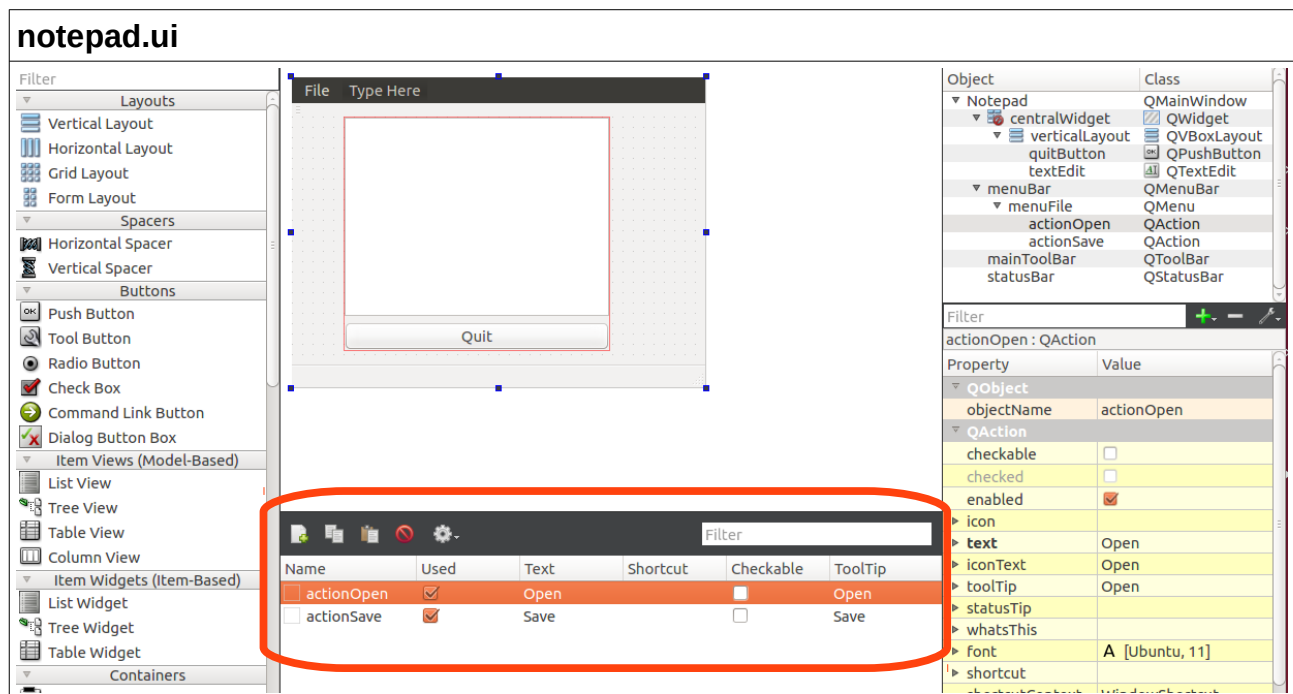
notepad.ui



Vamos adicionar as opções **Open** e **Save** nesse menu.

Wizard vai criar automaticamente ações correspondentes para esses itens (**QAction**).

Para adicionar a funcionalidade para itens de menu criados temos que conectar as ações com slots correspondentes. Para fazer isso temos que chamar o menu com o botão direito do mouse e escolher a opção **Go to slot > triggered()**



Qt Designer vai adicionar os seguintes private slots em **notepad.h** automaticamente:

- **on_actionOpen_triggered()**
- **on_actionSave_triggered()**

As funções correspondentes serão adicionados em **notepad.cpp**:

- **Notepad::on_actionOpen_triggered()**
- **Notepad::on_actionSave_triggered()**

Agora podemos completar o código com ações desejadas.

```

1  #ifndef NOTEPAD_H
2  #define NOTEPAD_H
3
4  #include <QMainWindow>
5
6  namespace Ui {
7  class Notepad;
8  }
9
10 class Notepad : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     explicit Notepad(QWidget *parent = 0);
16     ~Notepad();
17
18 private slots:
19     void on_quitButton_clicked();
20
21     void on_actionOpen_triggered();
22
23     void on_actionSave_triggered();
24
25 private:
26     Ui::Notepad *ui;
27 };
28
29 #endif // NOTEPAD_H
30

```

12) Abrindo e Salvando arquivo

Vamos aproveitar o dialogo padrão para arquivos [QFileDialog](#) e também vamos utilizar [QFile](#), [QMessageBox](#), e [QTextStream](#), Portanto precisamos incluir em nosso

notepad.cpp:

notepad.cpp

```
#include <QFileDialog>
#include <QFile>
#include <QMessageBox>
#include <QTextStream>
```

13) Para implementar a abertura do arquivo precisamos adicionar:

notepad.cpp

```
void Notepad::on_actionOpen_triggered()
{
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"),
    QString(),
        tr("Text Files (*.txt);;C++ Files (*.cpp *.h)"));

    if (!fileName.isEmpty()) {
        QFile file(fileName);
        if (!file.open(QIODevice::ReadOnly)) {
            QMessageBox::critical(this, tr("Error"), tr("Could not open
file"));
            return;
        }
        QTextStream in(&file);
        ui->textEdit->setText(in.readAll());
        file.close();
    }
}
```

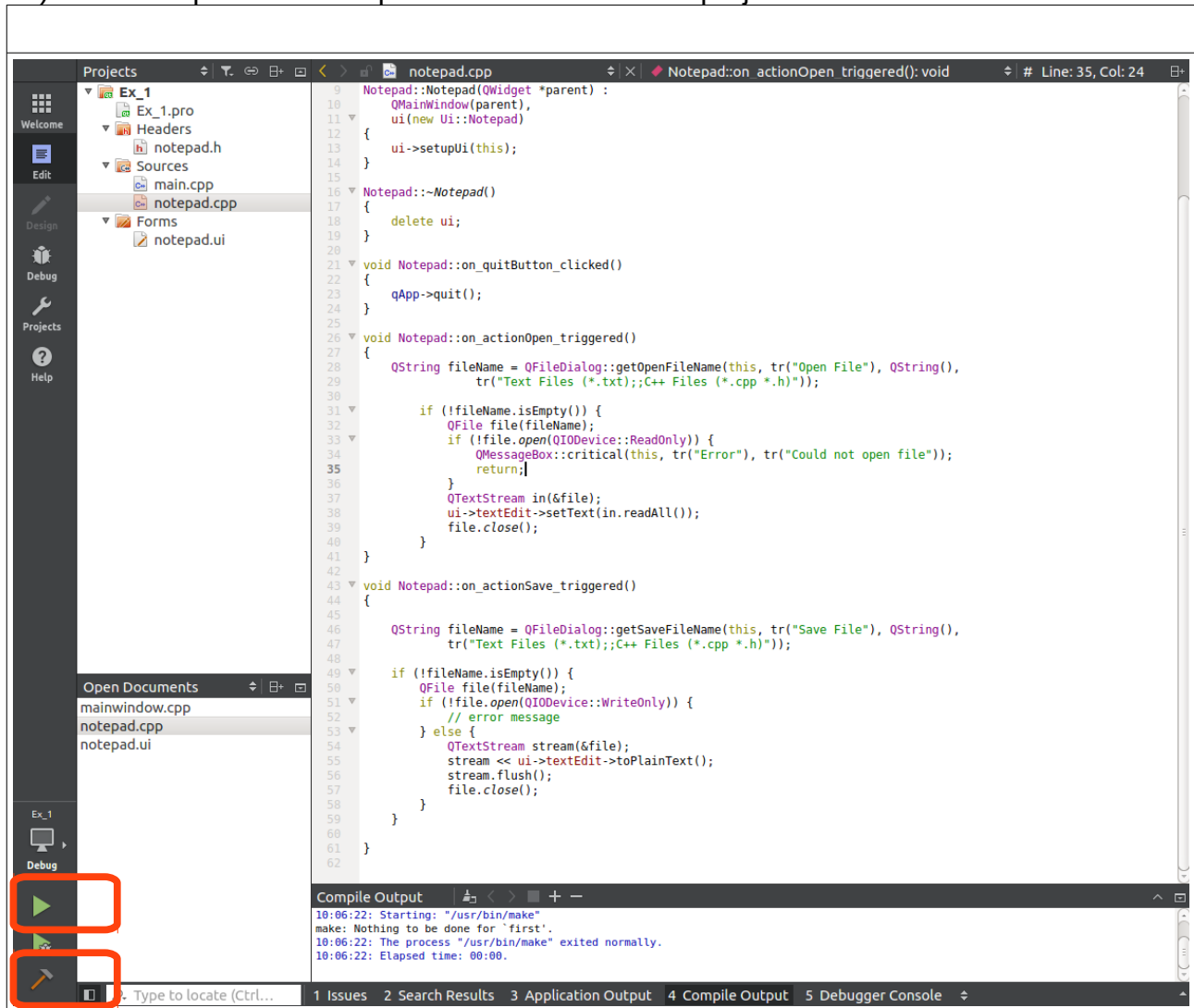
14) Para salvar arquivo precisamos adicionar:

notepad.cpp

```
void Notepad::on_actionSave_triggered()
{
    QString fileName = QFileDialog::getSaveFileName(this, tr("Save File"),
    QString(),
        tr("Text Files (*.txt);;C++ Files (*.cpp *.h)"));

    if (!fileName.isEmpty()) {
        QFile file(fileName);
        if (!file.open(QIODevice::WriteOnly)) {
            // error message
        } else {
            QTextStream stream(&file);
            stream << ui->textEdit->toPlainText();
            stream.flush();
            file.close();
        }
    }
}
}
```

15) Feito isso podemos compilar e executar o nosso projeto:



A aparência exata da janela do programa vai depender do sistema operacional, mas de forma geral vai ser similar a isso:

