

Introdução ao Framework Qt

Olá Mundo

- Danilo Almeida
- Bacharelado em Ciência da computação (UFV Florestal)
- Mestrado Em Ciência da computação (UFV)
- Analista de Desenvolvimento (DTI Digital)
- Lead Software Engineer (Cadence Design Systems)



O que é o Qt

- Framework para desenvolvimento de aplicações com interface gráfica



O que é o Qt

- Framework para desenvolvimento de aplicações com interface gráfica
- Desenvolvido em 1995 pela trolltech e atualmente mantido pelo *QT Project*



O que é o Qt

- Framework para desenvolvimento de aplicações com interface gráfica
- Desenvolvido em 1995 pela trolltech e atualmente mantido pelo *QT Project*
- Última versão estável 6.2 **LTS**



O que é o Qt

- Framework para desenvolvimento de aplicações com interface gráfica
- Desenvolvido em 1995 pela trolltech e atualmente mantido pelo *QT Project*
- Última versão estável 6.2 **LTS**
- É utilizado por várias empresas na construção de projetos para desktop, smartphone e sistemas embarcados

NOKIA
cā dence

Google

 **Toradex**
Swiss. Embedded. Computing.



Vantagens no uso do Qt

- Código multiplataforma

Vantagens no uso do Qt

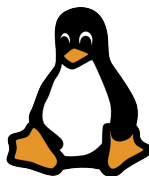
- Código multiplataforma
 - É possível compilar o mesmo código para diferentes tipos de sistema

ex:

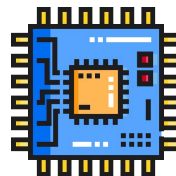
Preciso que a minha aplicação funcione em 3 ambientes diferentes



Windows



Linux



Sistema Embarcado

Vantagens no uso do Qt

```
using namespace std;
```

```
unsigned long long availableMemory(){  
    unsigned long long finalMemory;
```

⚠ To match this '{'

```
#ifdef LINUX
```



Implementação Linux

```
    QProcess p;  
    p.start("awk", QStringList() << "/MemTotal/ { print $2 }" << "/proc/meminfo");  
    p.waitForFinished();  
    finalMemory = p.readAllStandardOutput().toLong();  
    p.close();
```

```
#endif
```

```
#ifdef WINDOWS
```

```
    MEMORYSTATUSEX memory_status;  
    ZeroMemory(&memory_status, sizeof(MEMORYSTATUSEX));  
    memory_status.dwLength = sizeof(MEMORYSTATUSEX);  
    if (GlobalMemoryStatusEx(&memory_status)) {  
        DWORDLONG finalMemory = (memory_status.ullTotalPhys / (1024 * 1024));
```

```
#endif
```

```
#ifdef SISTEMA_EMBARCADO
```

```
    int value = 0;  
    int result = 0;  
    extern int *__brkval;  
    extern int __heap_start;  
    finalMemory = (unsigned long long)&value - ((unsigned long long)__brkval == 0 ? (unsigned long long)&__heap_start : (unsigned long long)__brkval);
```

```
#endif
```

```
    return finalMemory;
```

```
}
```

Vantagens no uso do Qt

```
using namespace std;
```

```
unsigned long long availableMemory(){  
    unsigned long long finalMemory;
```

⚠ To match this '{'

```
#ifdef LINUX
```

```
    QProcess p;  
    p.start("awk", QStringList() << "/MemTotal/ { print $2 }" << "/proc/meminfo");  
    p.waitForFinished();  
    finalMemory = p.readAllStandardOutput().toLong();  
    p.close();
```

```
#endif
```

```
#ifdef WINDOWS
```

```
    MEMORYSTATUSEX memory_status;  
    ZeroMemory(&memory_status, sizeof(MEMORYSTATUSEX));  
    memory_status.dwLength = sizeof(MEMORYSTATUSEX);  
    if (GlobalMemoryStatusEx(&memory_status)) {  
        DWORDLONG finalMemory = (memory_status.ullTotalPhys / (1024 * 1024));
```

```
#endif
```

```
#ifdef SISTEMA_EMBARCADO
```

```
    int value = 0;  
    int result = 0;  
    extern int *__brkval;  
    extern int __heap_start;  
    finalMemory = (unsigned long long)&value - ((unsigned long long)__brkval == 0 ? (unsigned long long)&__heap_start : (unsigned long long)__brkval);
```

```
#endif
```

```
    return finalMemory;
```

```
}
```

← Implementação Windows

Vantagens no uso do Qt

```
using namespace std;
```

```
unsigned long long availableMemory(){  
    unsigned long long finalMemory;
```

⚠ To match this '{'

```
#ifdef LINUX
```

```
    QProcess p;  
    p.start("awk", QStringList() << "/MemTotal/" << "print $2" << "/proc/meminfo");  
    p.waitForFinished();  
    finalMemory = p.readAllStandardOutput().toLong();  
    p.close();
```

```
#endif
```

```
#ifdef WINDOWS
```

```
    MEMORYSTATUSEX memory_status;  
    ZeroMemory(&memory_status, sizeof(MEMORYSTATUSEX));  
    memory_status.dwLength = sizeof(MEMORYSTATUSEX);  
    if (GlobalMemoryStatusEx(&memory_status)) {  
        DWORDLONG finalMemory = (memory_status.ullTotalPhys / (1024 * 1024));
```

```
#endif
```

```
#ifdef SISTEMA_EMBARCADO
```

```
    int value = 0;  
    int result = 0;  
    extern int *__brkval;  
    extern int __heap_start;  
    finalMemory = (unsigned long long)&value - ((unsigned long long)__brkval == 0 ? (unsigned long long)&__heap_start : (unsigned long long)__brkval);
```

```
#endif
```

```
    return finalMemory;
```

```
}
```

← Implementação Sistema embarcado Genérico

Vantagens no uso do Qt

- Com o Qt é possível encapsular este tipo de código conforme o tipo de arquitetura/ Sistema operacional para o qual você irá compilar o código
 - Boa parte das coisas que precisamos e que são implementadas de formas diferentes dependendo do sistema operacional ou hardware, podem ser utilizadas sem a necessidade de fazer esse tipo de “Gambiarra”
- Open Source

Vantagens no uso do Qt

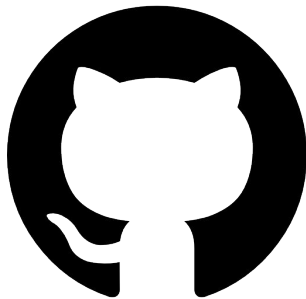
- Com o Qt é possível encapsular este tipo de código conforme o tipo de arquitetura/ Sistema operacional para o qual você irá compilar o código
 - Boa parte das coisas que precisamos e que são implementadas de formas diferentes dependendo do sistema operacional ou hardware, podem ser utilizadas sem a necessidade de fazer esse tipo de “Gambiarra”
- “Open Source”



Antes de começar

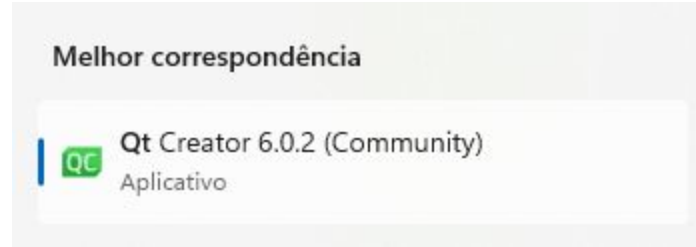
- Conferir se todas as máquinas possuem o ambiente de desenvolvimento instalado
- Qt Creator + Kit de desenvolvimento Desktop Qt 6.4.0 MinGW 64-bit
- Clonar o repositório:

<https://github.com/danilo94/MiniCursoQt>



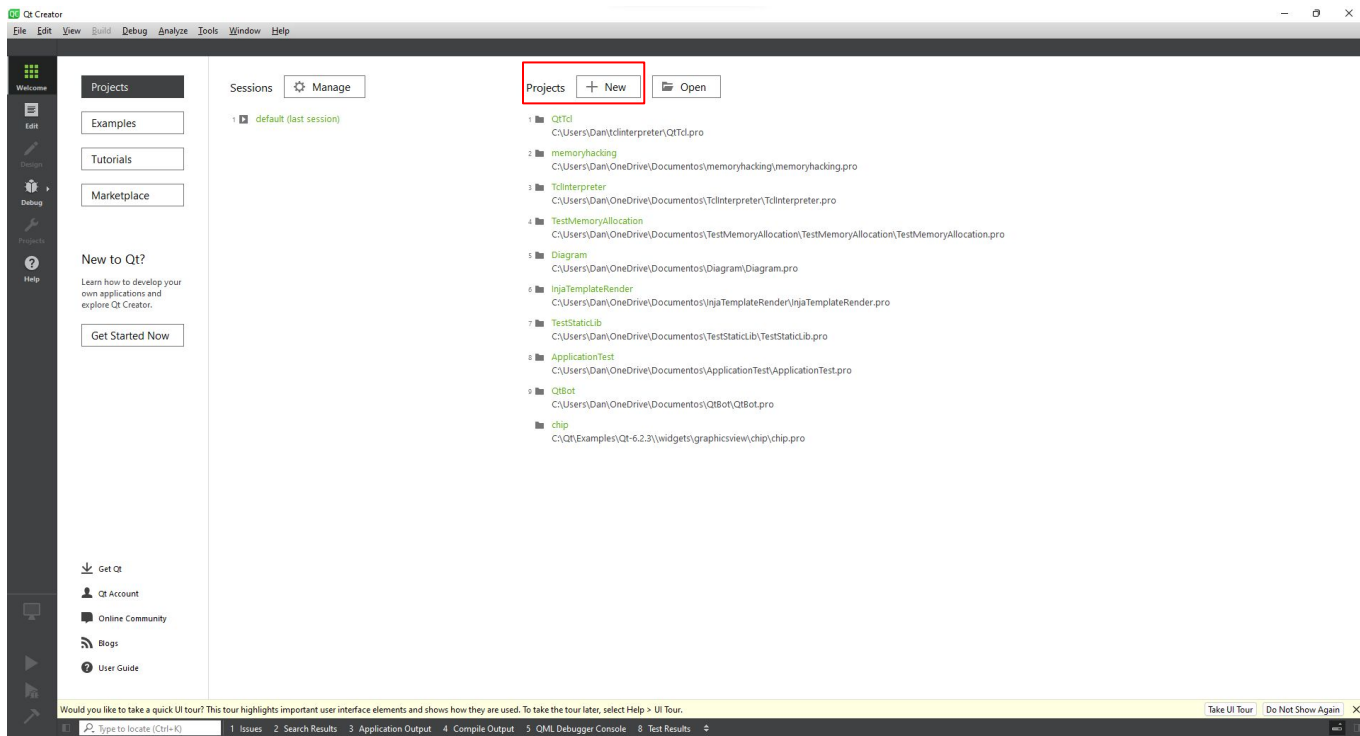
Criando seu primeiro projeto no Qt

1. Abra o Qt Creator na sua máquina



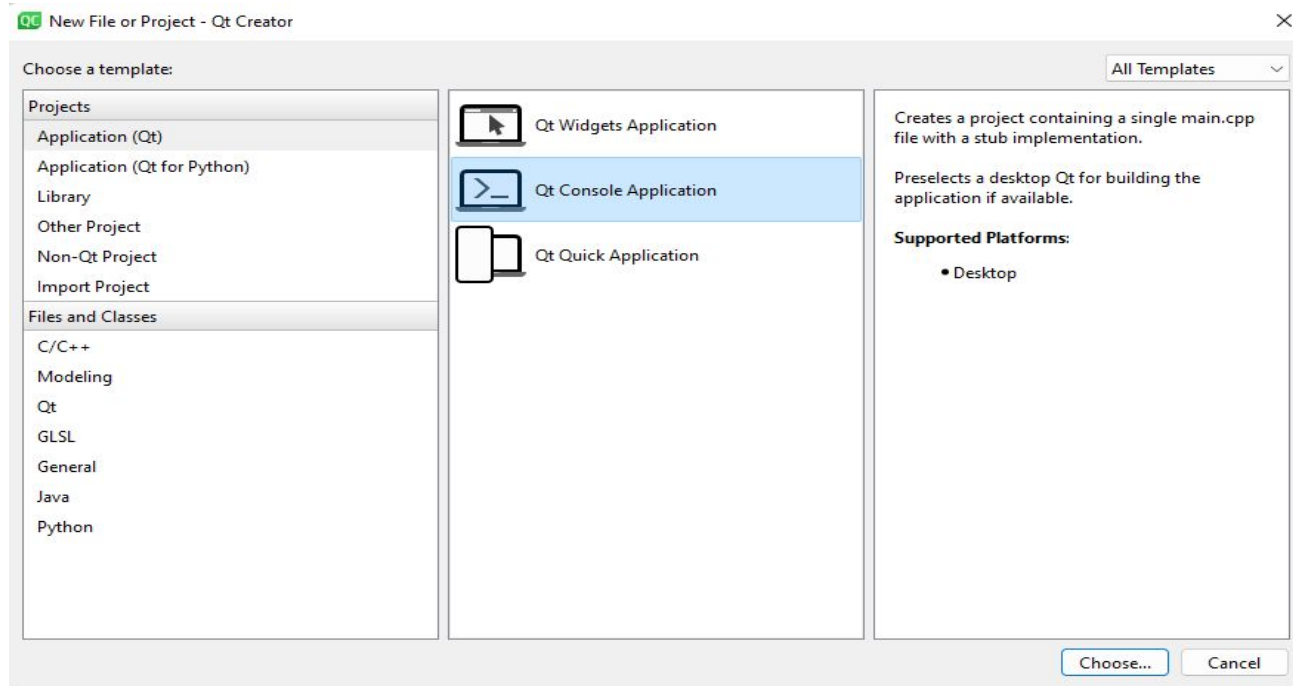
Criando seu primeiro projeto no Qt

2. Na aba Projects selecione a opção Projects: **New**



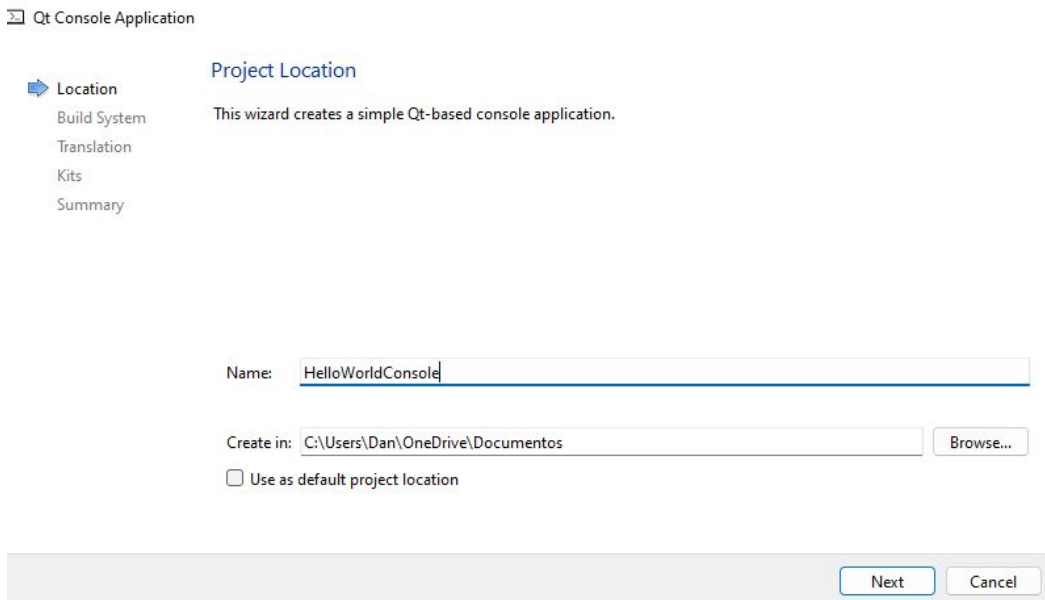
Criando seu primeiro projeto no Qt

3. dentro da opção New File or Project selecione **Qt Console Application**



Criando seu primeiro projeto no Qt

4. Nesta etapa iremos definir o nome do nosso projeto, esse primeiro exemplo iremos chamar de **HelloWorldConsole**



Qt Console Application

Location

- Build System
- Translation
- Kits
- Summary

Project Location

This wizard creates a simple Qt-based console application.

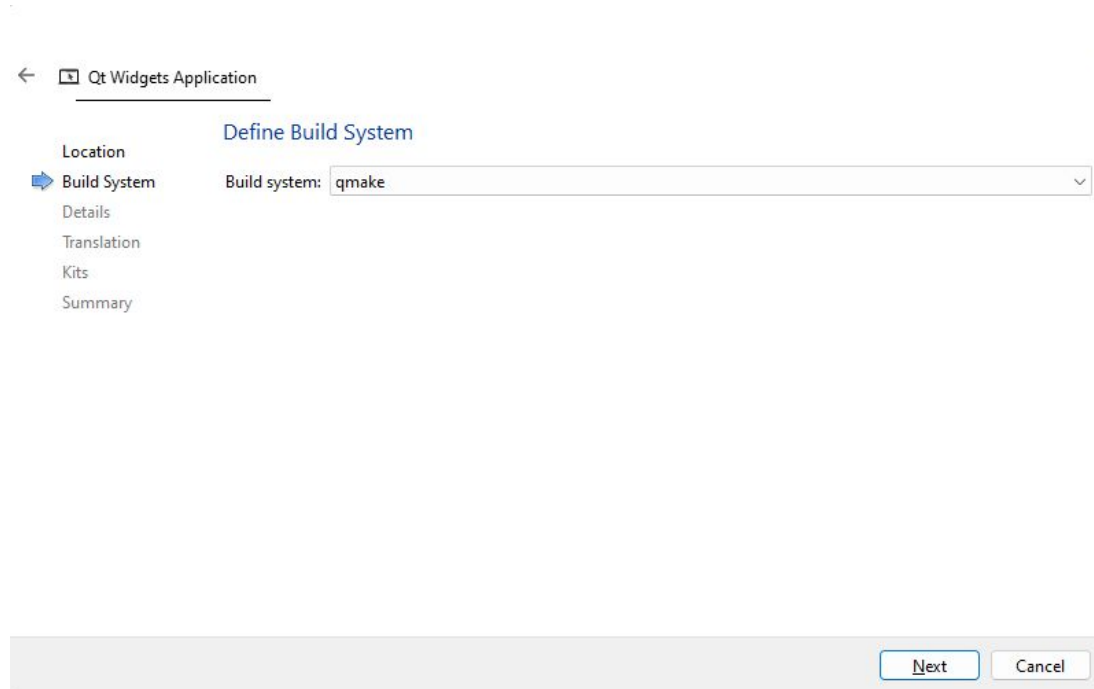
Name:

Create in:

☐ Use as default project location

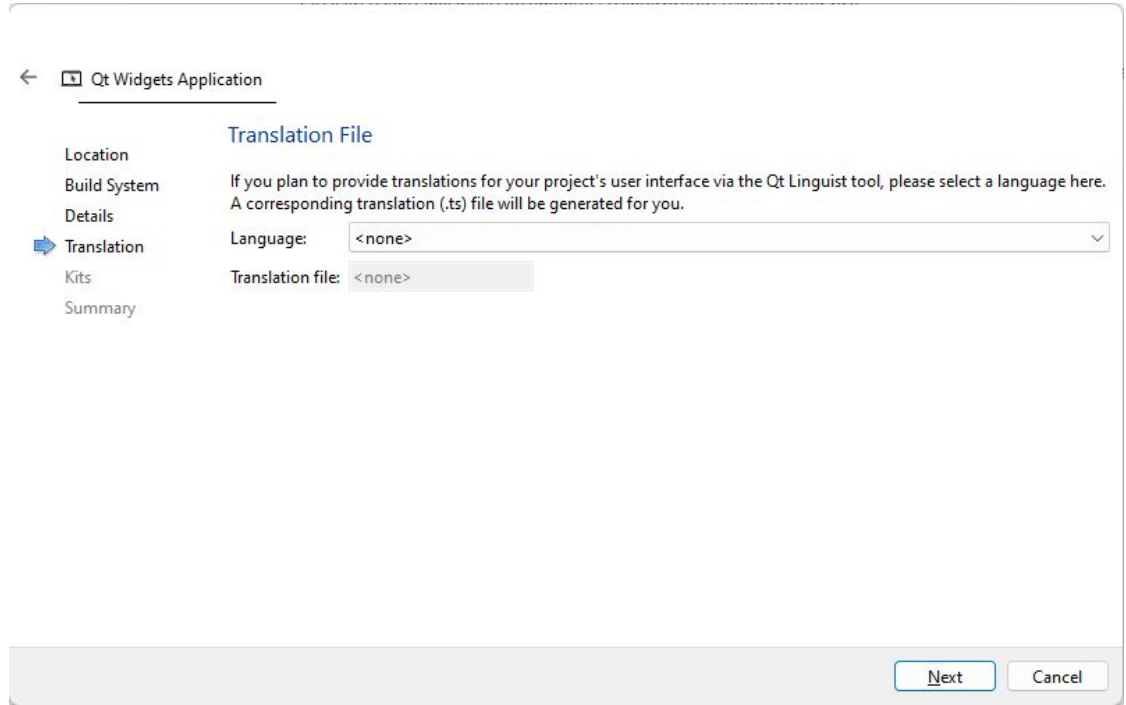
Criando seu primeiro projeto no Qt

5. Utilizaremos a opção de build padrão do Qt, neste caso o **qmake**



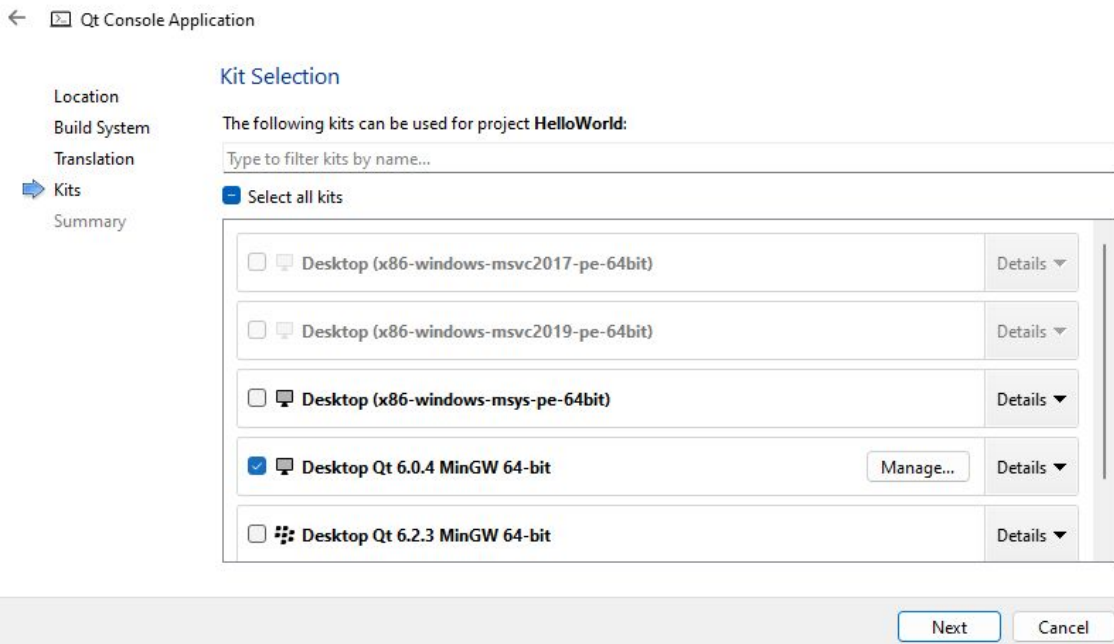
Criando seu primeiro projeto no Qt

6. O Qt possui suporte a arquivos de tradução, porém para este curso iremos manter a opção none



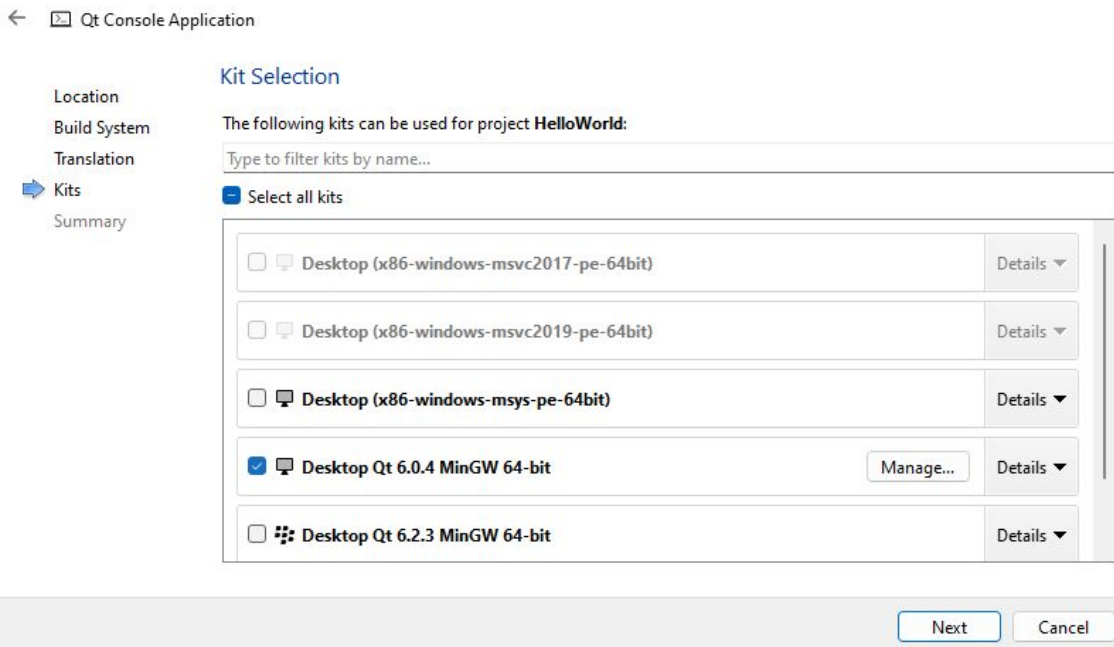
Criando seu primeiro projeto no Qt

7. Na opção Kits, iremos selecionar a versão Desktop **Qt 6.0.4 MinGW 64-bit**



Criando seu primeiro projeto no Qt

7. Na opção Kits, iremos selecionar a versão Desktop **Qt 6.0.4 MinGW 64-bit**

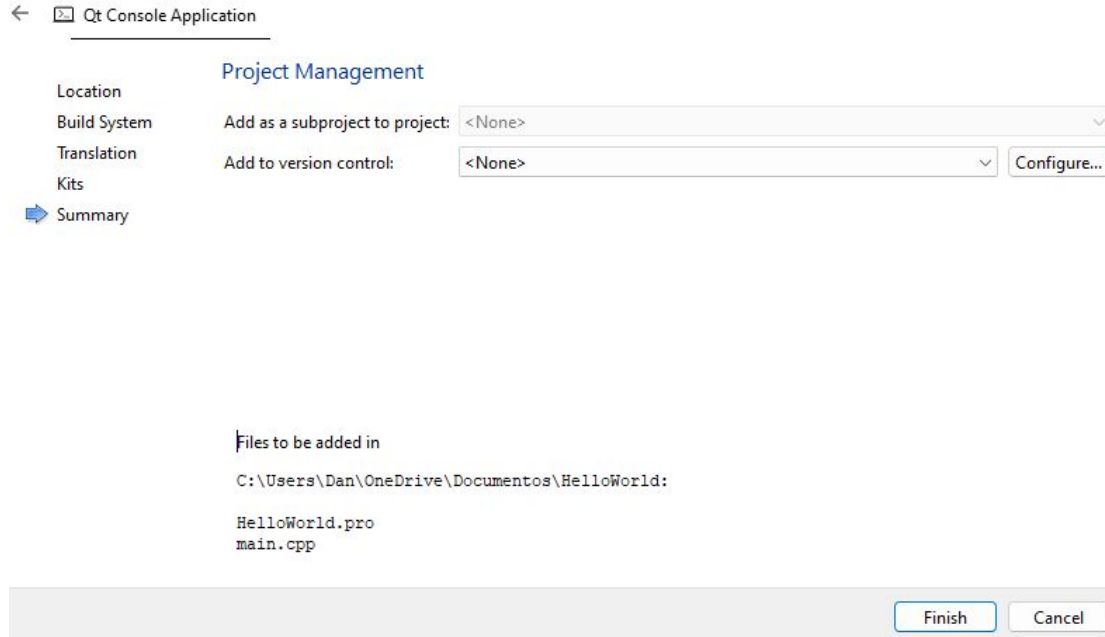


* O MinGW é um conjunto de ferramentas do GNU portadas para o Windows

- G++
- GCC
- Etc..

Criando seu primeiro projeto no Qt

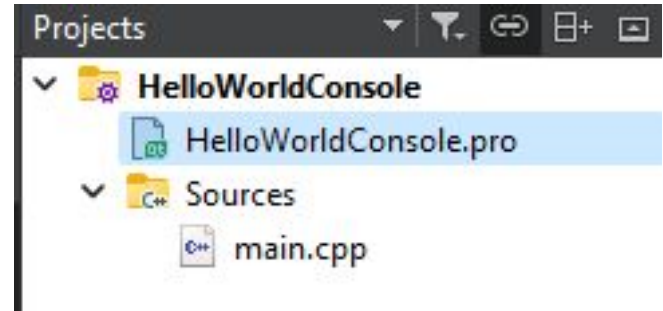
8. Por fim selecionaremos a opção Finish para criação do projeto



Explorando os arquivos gerados

Ao pressionar o Botão Finish, serão criados dois arquivos denominados:

1. **HelloWorldConsole.pro**
2. **Main.cpp**



Explorando os arquivos gerados

- **HelloWorldConsole.pro**

Arquivo gerado pelo Qt que é utilizado para definição de todas as regras a serem seguidas no processo de compilação

Explorando os arquivos gerados

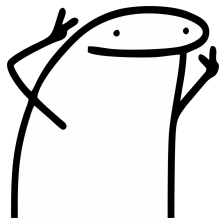
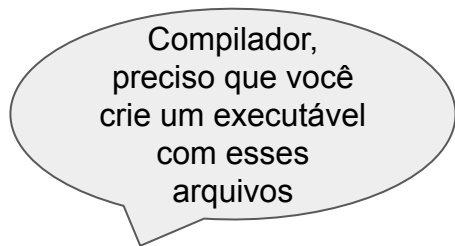
- **HelloWorldConsole.pro**

Arquivo gerado pelo Qt que é utilizado para definição de todas as regras a serem seguidas no processo de compilação



Como uma aplicação C/C++ é compilada

1. Precisamos informar para o compilador tudo que deve ser compilado e linkado para construção do nosso projeto



Eu



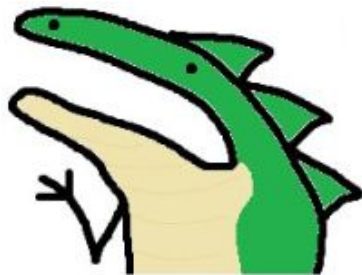
main.cpp



lib.cpp



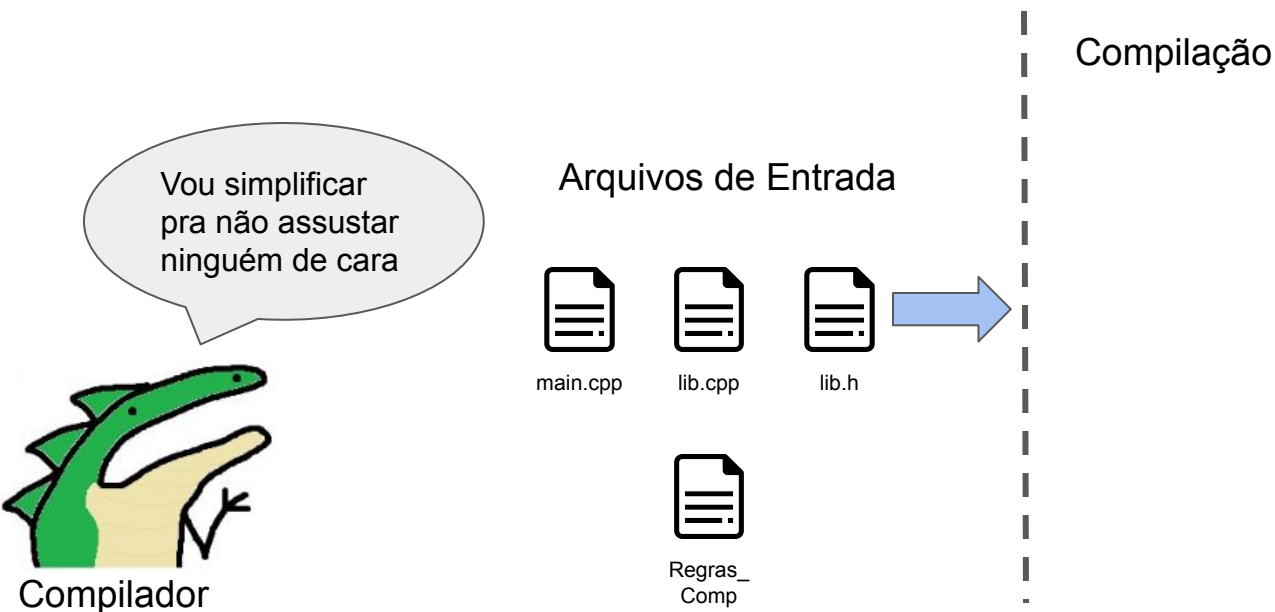
lib.h



Compilador

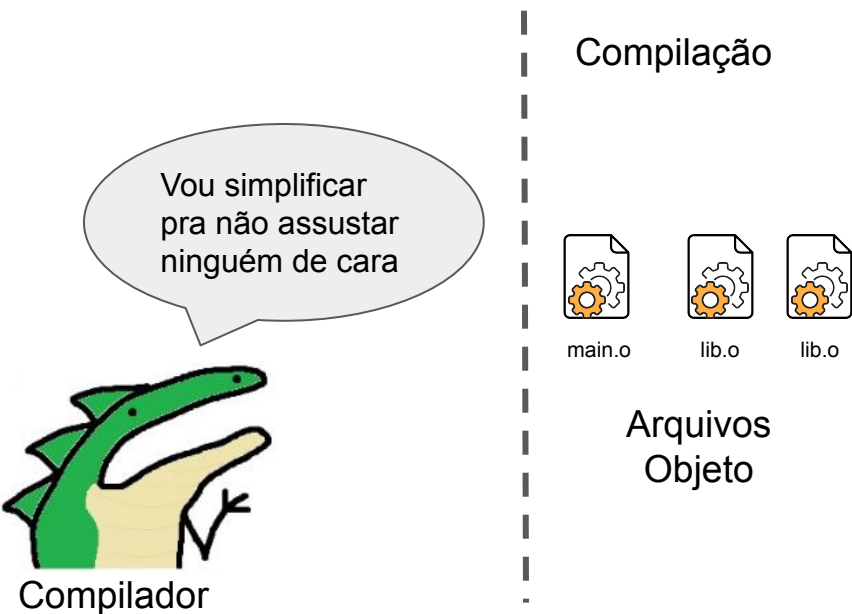
Como uma aplicação C/C++ é compilada

2. Primeiramente todos os arquivos e regras de compilação que o compilador precisa utilizar são repassados



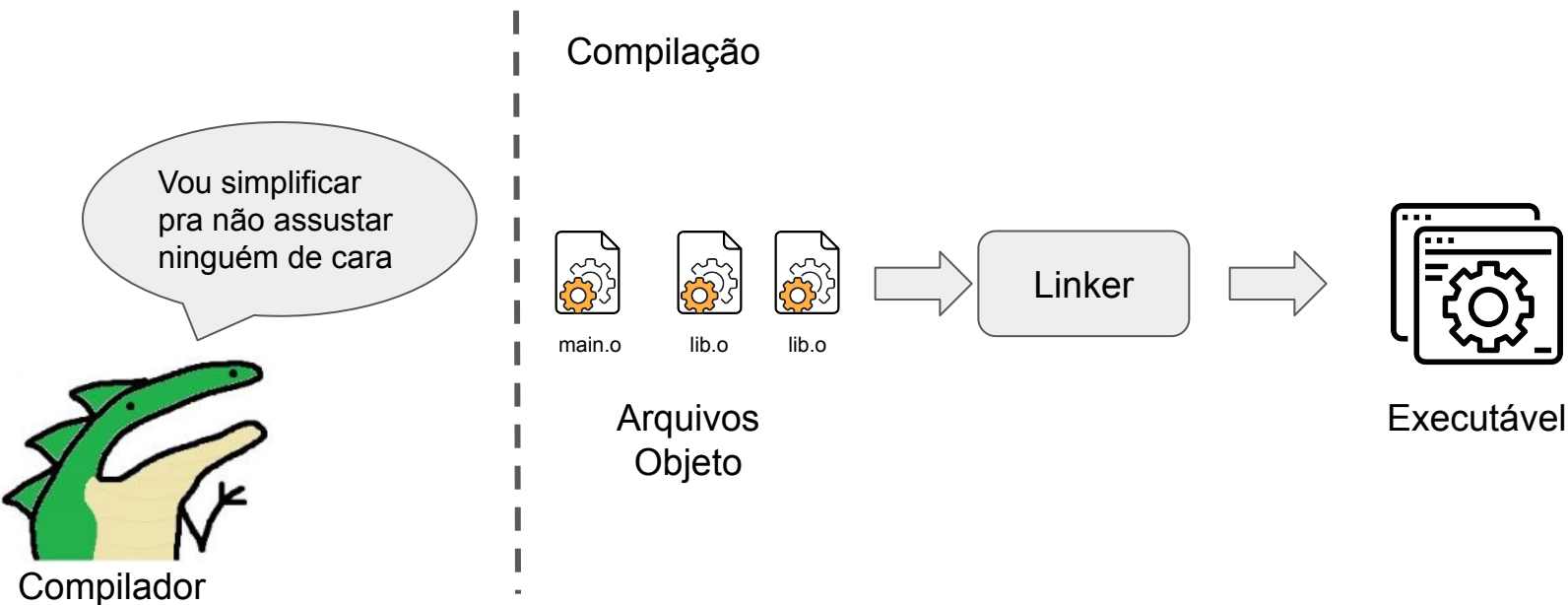
Como uma aplicação C/C++ é compilada

3. O compilador por sua vez vai fazer todo o processo de análise do código e gerar os arquivos de objeto

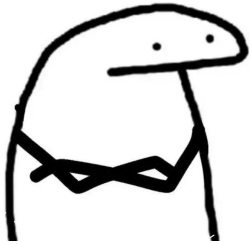


Como uma aplicação C/C++ é compilada

4. Com todos os objetos compilados o processo de linkagem é iniciado e todos os arquivos são combinados em um executável



Como uma aplicação C/C++ é compilada



Eu



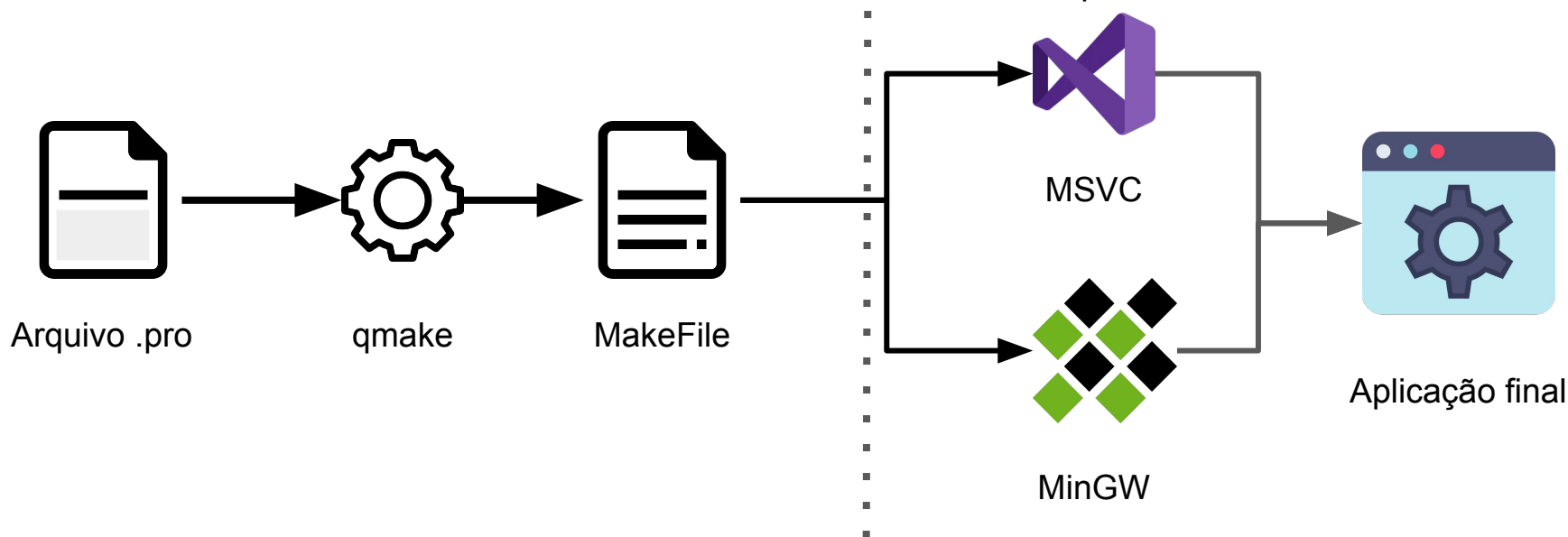
Compilador

Explorando os arquivos gerados

- **HelloWorldConsole.pro**

Então este arquivo será utilizado como um guia para a geração do seu executável

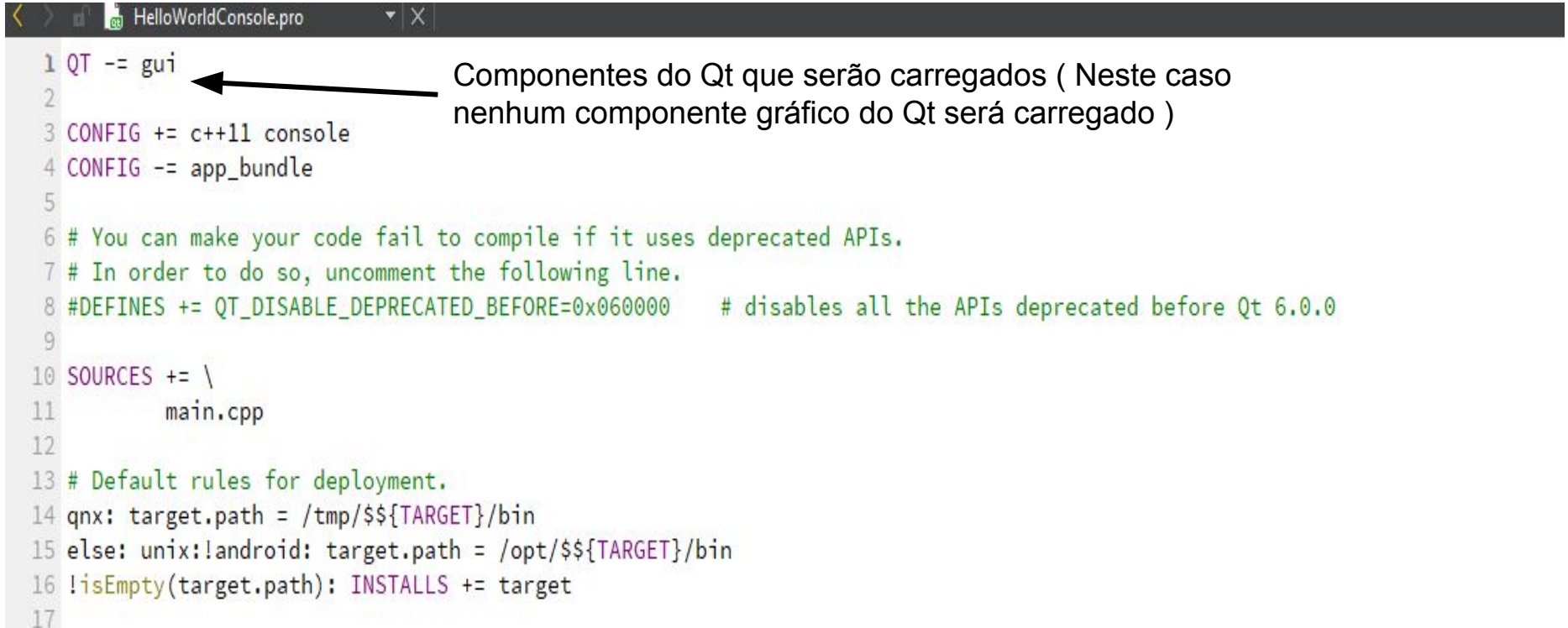
Compiladores



Exemplo de arquivo .pro

```
1 QT -= gui
2
3 CONFIG += c++11 console
4 CONFIG -= app_bundle
5
6 # You can make your code fail to compile if it uses deprecated APIs.
7 # In order to do so, uncomment the following line.
8 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0
9
10 SOURCES += \
11     main.cpp
12
13 # Default rules for deployment.
14 qnx: target.path = /tmp/${TARGET}/bin
15 else: unix:!android: target.path = /opt/${TARGET}/bin
16 isEmpty(target.path): INSTALLS += target
17
```

Exemplo de arquivo .pro



```
1 QT -= gui
2
3 CONFIG += c++11 console
4 CONFIG -= app_bundle
5
6 # You can make your code fail to compile if it uses deprecated APIs.
7 # In order to do so, uncomment the following line.
8 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0
9
10 SOURCES += \
11     main.cpp
12
13 # Default rules for deployment.
14 qnx: target.path = /tmp/${TARGET}/bin
15 else: unix:!android: target.path = /opt/${TARGET}/bin
16 isEmpty(target.path): INSTALLS += target
17
```

Componentes do Qt que serão carregados (Neste caso nenhum componente gráfico do Qt será carregado)

Exemplo de arquivo .pro

```
< > HelloWorldConsole.pro X
1 QT -= gui
2
3 CONFIG += c++11 console
4 CONFIG -= app_bundle
5
6 # You can make your code fail to compile if it uses deprecated APIs.
7 # In order to do so, uncomment the following line.
8 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0
9
10 SOURCES += \
11     main.cpp
12
13 # Default rules for deployment.
14 qnx: target.path = /tmp/${TARGET}/bin
15 else: unix:!android: target.path = /opt/${TARGET}/bin
16 isEmpty(target.path): INSTALLS += target
17
```

Configurações

- Versão do C++ utilizada
- se a aplicação deverá ser aberta em modo console

Exemplo de arquivo .pro





```

1 QT -= gui
2
3 CONFIG += c++11 console
4 CONFIG -= app_bundle
5
6 # You can make your code fail to compile if it uses deprecated APIs.
7 # In order to do so, uncomment the following line.
8 #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0
9
10 SOURCES += \
11     main.cpp
12
13 # Default rules for deployment.
14 qnx: target.path = /tmp/${TARGET}/bin
15 else: unix:!android: target.path = /opt/${TARGET}/bin
16 isEmpty(target.path): INSTALLS += target
17
```

Sources

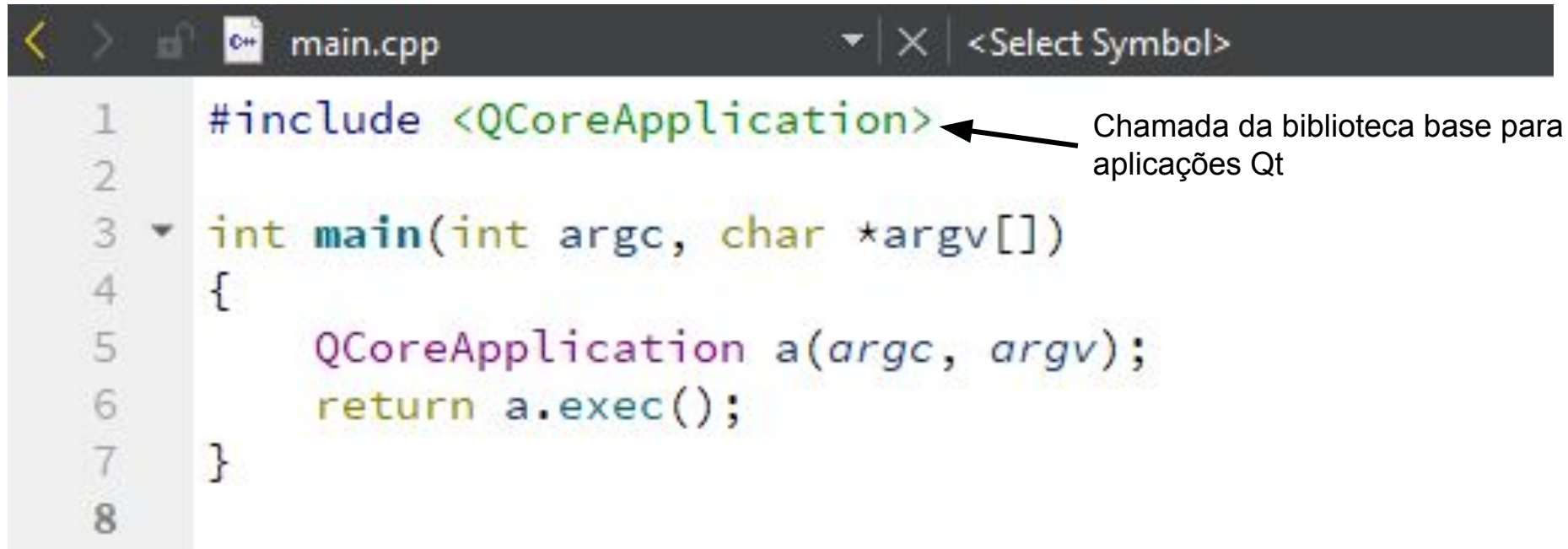
- Arquivos que devem ser lidos na hora de gerar o arquivo compilado final




Exemplo de Arquivo main.cpp

```
< >   main.cpp   <Select Symbol>

1  #include <QCoreApplication>
2
3  ▼ int main(int argc, char *argv[])
4  {
5      QCoreApplication a(argc, argv);
6      return a.exec();
7  }
8
```

Exemplo de Arquivo main.cpp

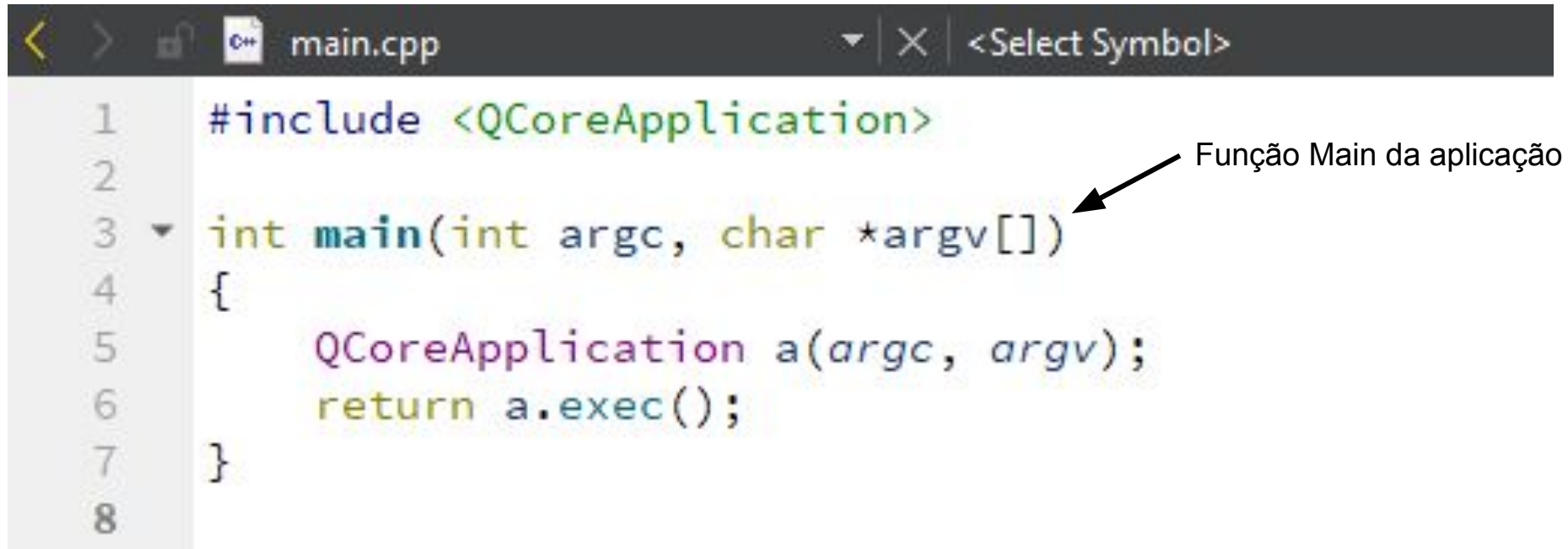


```
< >  main.cpp   <Select Symbol>
```

```
1  #include <QCoreApplication>
2
3  int main(int argc, char *argv[])
4  {
5      QCoreApplication a(argc, argv);
6      return a.exec();
7  }
8
```

Chamada da biblioteca base para aplicações Qt

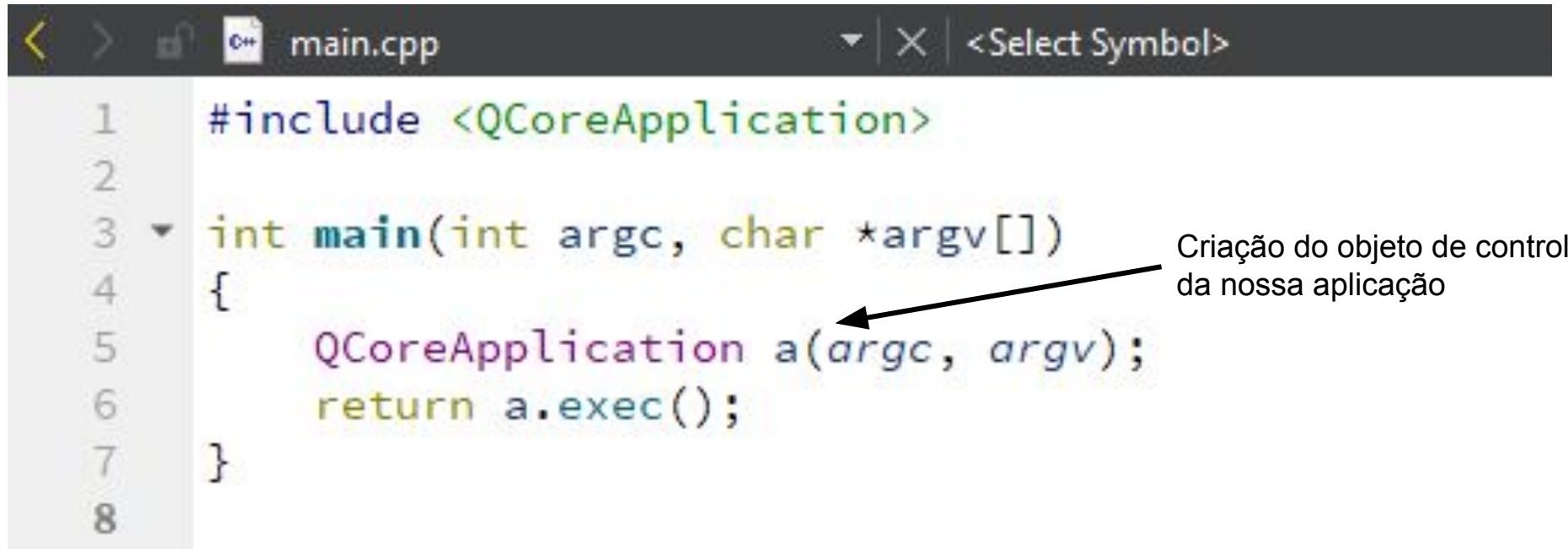
Exemplo de Arquivo main.cpp



```
< > main.cpp <Select Symbol>
1  #include <QCoreApplication>
2
3  int main(int argc, char *argv[])
4  {
5      QCoreApplication a(argc, argv);
6      return a.exec();
7  }
8
```

Função Main da aplicação

Exemplo de Arquivo main.cpp



```
< > main.cpp <Select Symbol>
1  #include <QCoreApplication>
2
3  int main(int argc, char *argv[])
4  {
5      QCoreApplication a(argc, argv);
6      return a.exec();
7  }
8
```

Criação do objeto de control
da nossa aplicação

Exemplo de Arquivo main.cpp

```
< > main.cpp <Select Symbol>
1  #include <QCoreApplication>
2
3  int main(int argc, char *argv[])
4  {
5      QCoreApplication a(argc, argv);
6      return a.exec();
7  }
8
```

← Inicialização do event loop da aplicação

Exemplo de Arquivo main.cpp

Adicione a seguinte linha na linha 6

```
qDebug() << "Hello World";
```

O que é um Event Loop ?

O Qt segue um padrão de funcionamento orientado a eventos, mas na prática como isso funciona ?

O que é um Event Loop ?

Imagine o seguinte trecho de código:

```
while (true){  
    atualizarInterfaceGrafica();  
    processarEventos();  
    aguardarNovosEventos();  
}
```

O que é um Event Loop ?

1. O primeiro método atualiza todos os componentes gráficos que estão visíveis na tela

```
while (true){  
    atualizarInterfaceGrafica();  
    processarEventos();  
    aguardarNovosEventos();  
}
```

O que é um Event Loop ?

2. O segundo método processa todos os eventos que foram registrados na aplicação

```
while (true){  
    atualizarInterfaceGrafica();  
    processarEventos();  
    aguardarNovosEventos();  
}
```

O que é um Event Loop ?

Mas o que é um evento ?



O que é um Event Loop ?

Podemos definir um evento como toda ação que está relacionada a um agente externo ao nosso fluxo de fora ao Event loop

O que é um Event Loop ?

Exemplos:

1. Interações do usuário com a interface gráfica
 - a. Pressionar um botão
 - b. Mover o mouse na tela

O que é um Event Loop ?

Exemplos:

1. Interações do usuário com a interface gráfica
 - a. Pressionar um botão
 - b. Mover o mouse na tela
2. Comunicação
 - a. Receber uma mensagem via Socket
 - b. Receber uma mensagem de outro processo

O que é um Event Loop ?

Exemplos:

1. Interações do usuário com a interface gráfica
 - a. Pressionar um botão
 - b. Mover o mouse na tela
2. Comunicação
 - a. Receber uma mensagem via Socket
 - b. Receber uma mensagem de outro processo
3. Temporais
 - a. Timeout de um temporizador

O que é um Event Loop ?

3. Já o terceiro método por sua vez aguardará até que novos eventos aconteçam

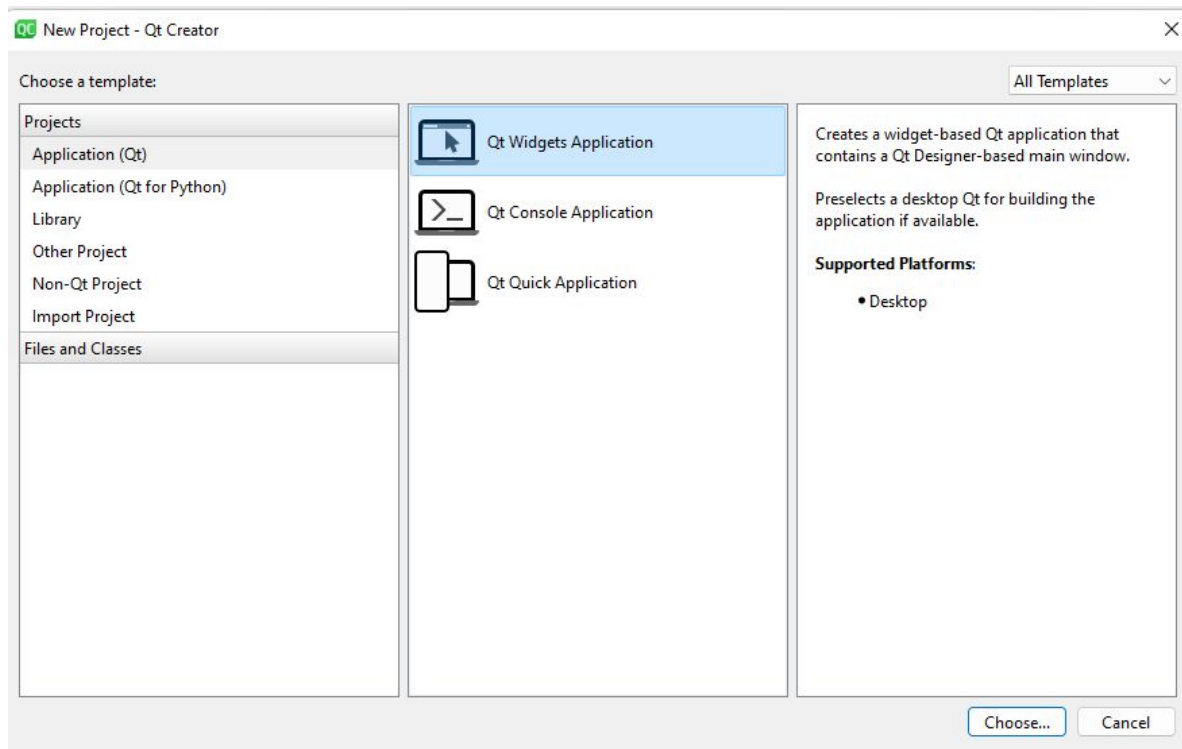
```
while (true){  
    atualizarInterfaceGrafica();  
    processarEventos();  
    aguardarNovosEventos();  
}
```

Criando seu primeiro projeto com interface gráfica no Qt

Agora vamos criar o nosso projeto utilizando uma interface gráfica

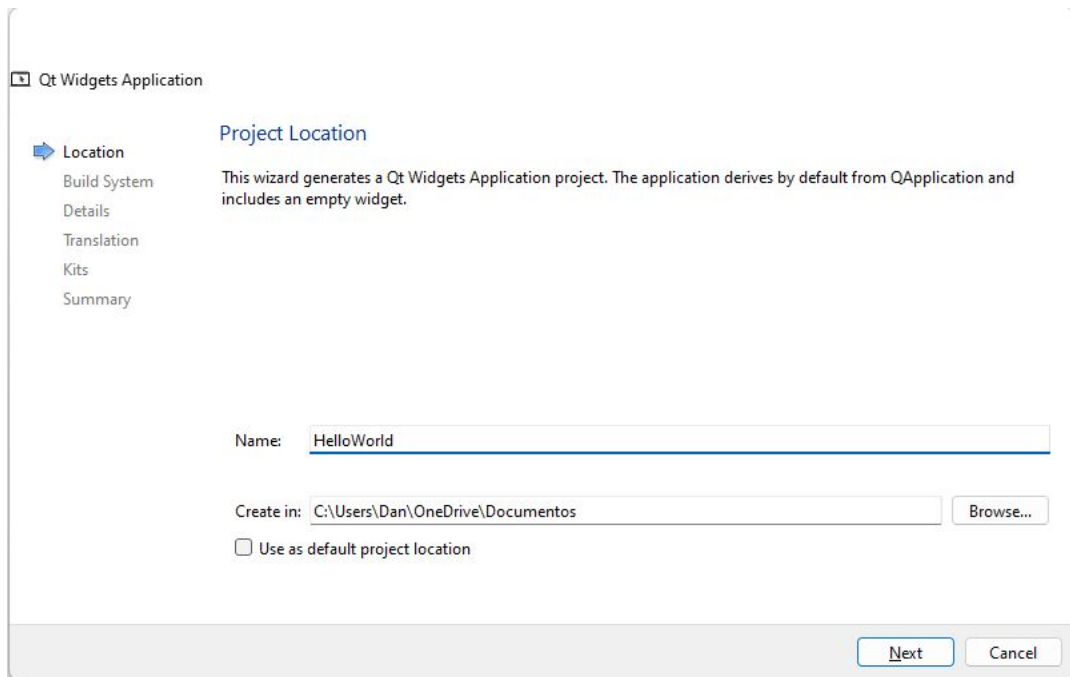
Criando seu primeiro projeto com interface gráfica no Qt

1. Dentro de New Project, selecione a opção Qt Widgets Application



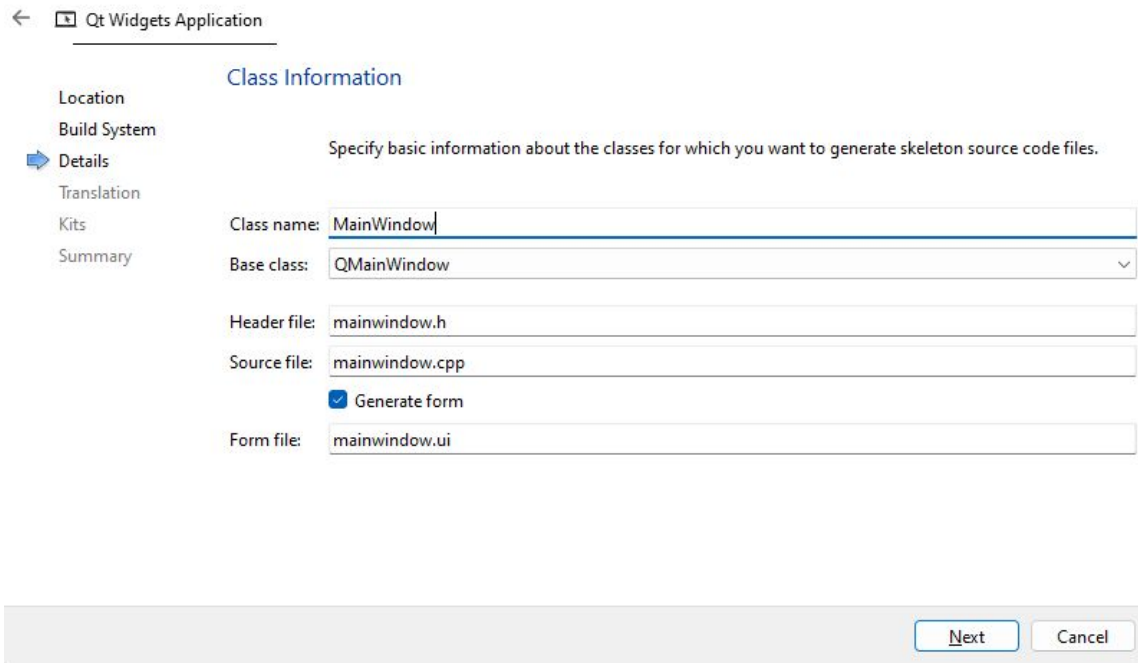
Criando seu primeiro projeto com interface gráfica no Qt

2. Iremos definir o nome do nosso projeto inicial como **HelloWorld**



Criando seu primeiro projeto com interface gráfica no Qt

3. Nesta etapa definimos os nomes iniciais que iremos utilizar para construir o projeto base da aplicação



The image shows the 'Qt Widgets Application' dialog box in Qt Creator, specifically the 'Class Information' tab. On the left, a sidebar lists the steps: Location, Build System, Details (highlighted with a blue arrow), Translation, Kits, and Summary. The main area is titled 'Class Information' and contains the instruction: 'Specify basic information about the classes for which you want to generate skeleton source code files.' Below this, there are several input fields: 'Class name:' with 'MainWindow' entered; 'Base class:' with 'QMainWindow' selected from a dropdown; 'Header file:' with 'mainwindow.h' entered; 'Source file:' with 'mainwindow.cpp' entered; a checked checkbox for 'Generate form'; and 'Form file:' with 'mainwindow.ui' entered. At the bottom right, there are 'Next' and 'Cancel' buttons.

← Qt Widgets Application

Class Information

Location
Build System
Details
Translation
Kits
Summary

Specify basic information about the classes for which you want to generate skeleton source code files.

Class name: MainWindow

Base class: QMainWindow

Header file: mainwindow.h

Source file: mainwindow.cpp

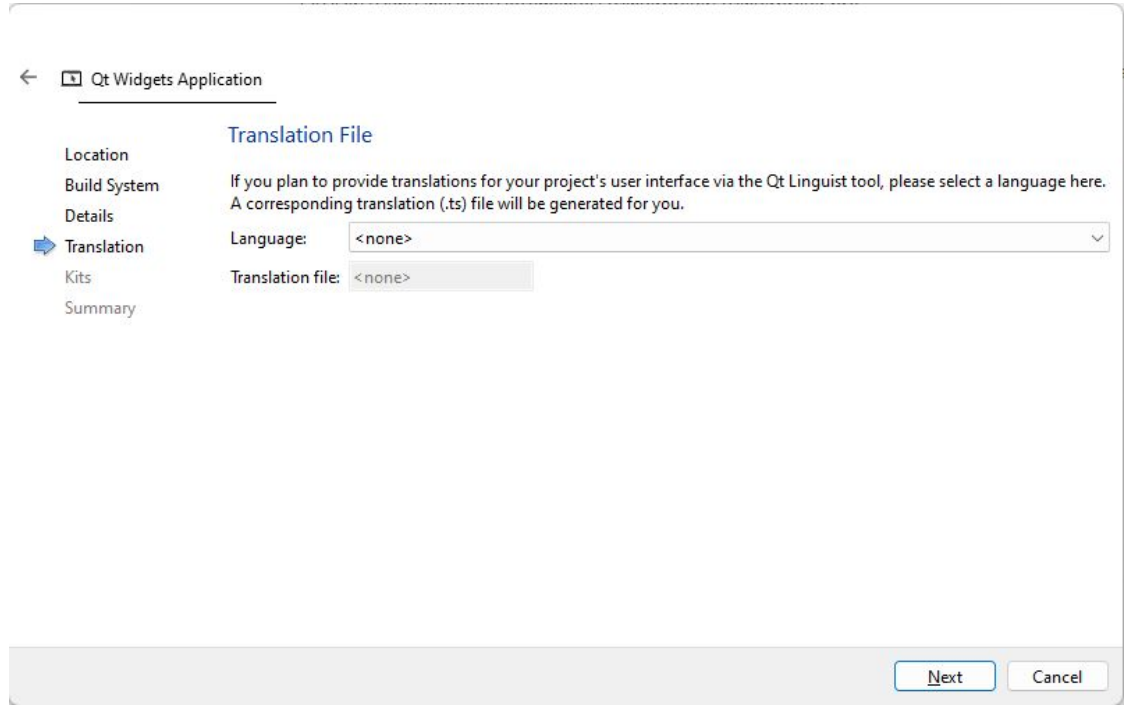
☒ Generate form

Form file: mainwindow.ui

Next Cancel

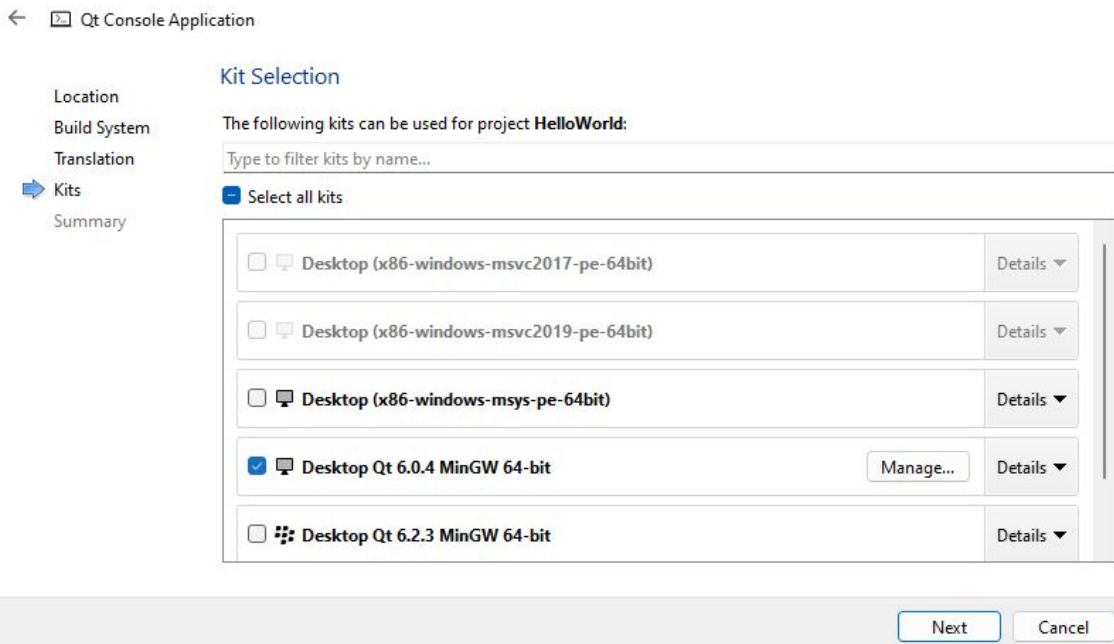
Criando seu primeiro projeto com interface gráfica no Qt

4. O Qt possui suporte a arquivos de tradução, porém para este curso iremos manter a opção none



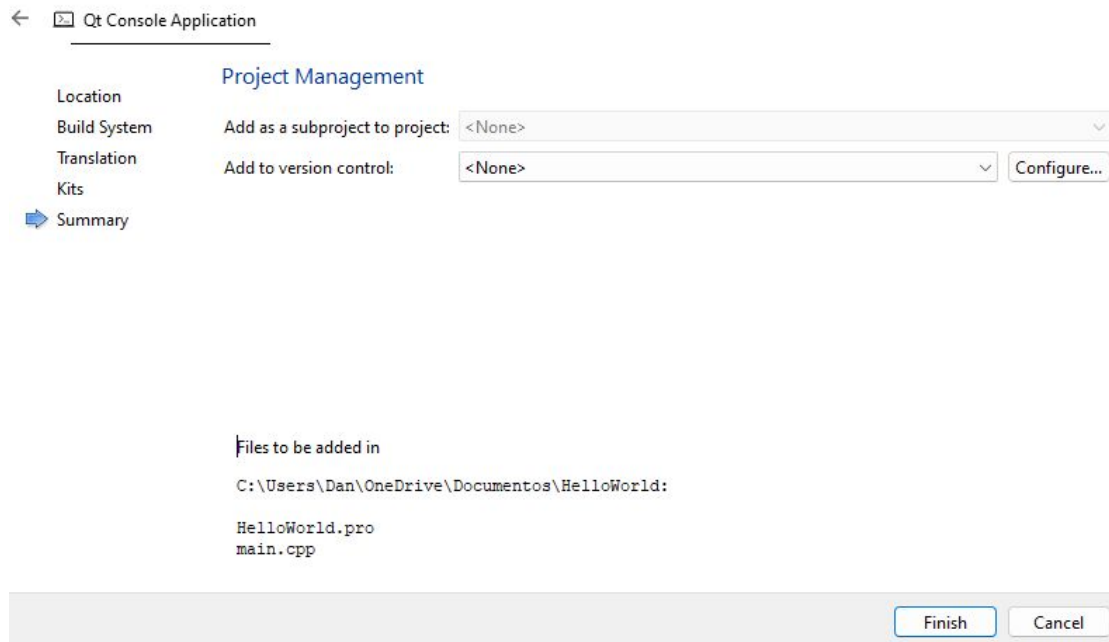
Criando seu primeiro projeto com interface gráfica no Qt

5. Na opção Kits, iremos selecionar a versão Desktop **Qt 6.0.4 MinGW 64-bit**



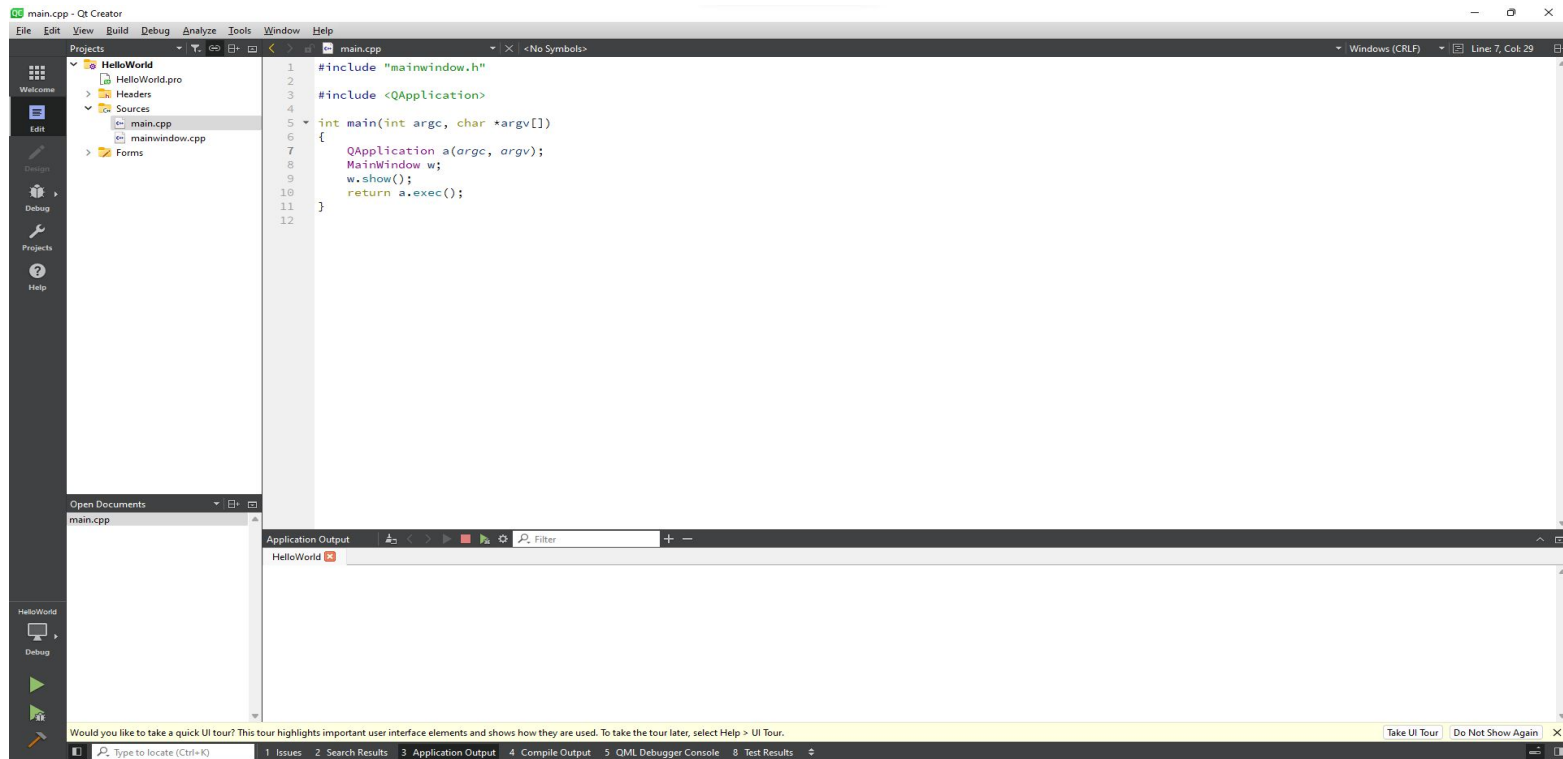
Criando seu primeiro projeto com interface gráfica no Qt

6. Por fim selecionaremos a opção Finish para criação do projeto




Criando seu primeiro projeto com interface gráfica no Qt

10. Ao fim de tudo teremos um projeto criado, e alguns arquivos de esqueleto



Criando seu primeiro projeto com interface gráfica no Qt


▼ 02-PrimeiroFormularioQt [main]

 02-PrimeiroFormularioQt.pro

▼ Headers

 mainwindow.h

▼ Sources

 main.cpp

 mainwindow.cpp

▼ Forms

 mainwindow.ui

Sistema de interação entre objetos

No Qt, quando precisamos que objetos troquem informações entre si, utilizamos o mecanismo de Signal/Slot

Sistema de interação entre objetos

No Qt, quando precisamos que objetos troquem informações entre si, utilizamos o mecanismo de Signal/Slot

- Um objeto qt é capaz de emitir “sinais” sempre que precisar avisar outro objeto que alguma mudança aconteceu
- Essa estrutura de comunicação é interessante pois facilita o desacoplamento entre os objetos, dividindo as suas responsabilidades

Sistema de interação entre objetos

No Qt, quando precisamos que objetos troquem informações entre si, utilizamos o mecanismo de Signal/Slot

- Para que isso seja possível, uma classe precisa seguir os seguintes critérios:
 - Ser herdeira da classe QObject
 - A macro Q_OBJECT deve existir no início da criação da classe
 - Todos os métodos criados como signal, não devem possuir uma implementação
 - Todos os métodos criados como slot, devem possuir uma implementação
 - Ao realizar uma conexão entre um signal e um slot, os mesmos devem possuir a mesma assinatura de método (caso contrário a conexão não irá funcionar)

Sistema de interação entre objetos

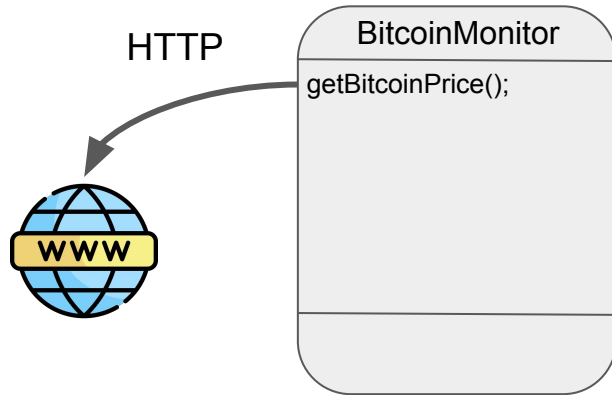
- Exemplo:

Abrir prática 03 - Criando seu primeiro Signal/Slot

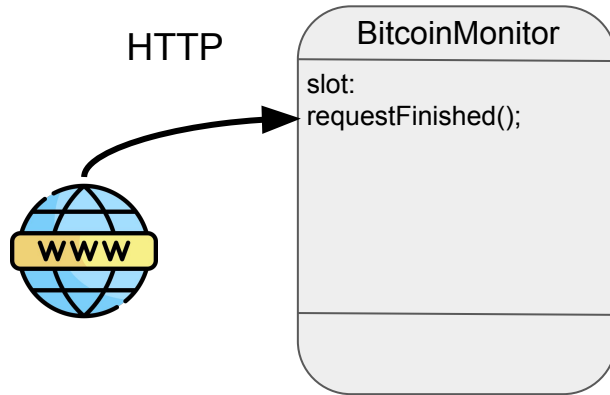
Sistema de interação entre objetos

- Exemplo 2:
 - Uma aplicação que monitora a variação do preço do bitcoin em intervalos regulares de tempo pode ser dividida em duas camadas:
 - **BitCoinMonitor** - Objeto responsável por realizar as requisições em uma API externa e retornar esses dados para a GUI
 - **MainWindow** - Objeto responsável por receber as informações do **BitCoinMonitor** e renderizar os dados atualizados na tela

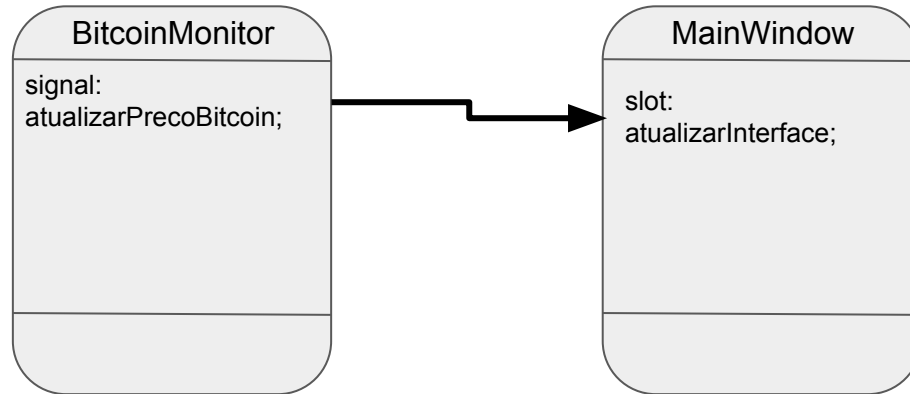
Sistema de interação entre objetos



Sistema de interação entre objetos



Sistema de interação entre objetos



Sistema de interação entre objetos

- Exemplo:

Abrir prática 04 - **SignalSlotBitcoinPrice**

Ciclo de vida de objetos Qt

- Linguagens como C/C++ não possuem mecanismos de gerenciamento automático de memória

Ciclo de vida de objetos Qt

- Linguagens como C/C++ não possuem mecanismos de gerenciamento automático de memória
- Como resultado, se o fluxo de um código não for muito bem pensado, podemos eventualmente ter problemas com relação a memória não alocada corretamente

Ciclo de vida de objetos Qt

- Vulgo Vazamento de memória



Ciclo de vida de objetos Qt

- No Qt não é muito diferente, porém para gerenciar o fluxo de criação e destruição de objetos no Qt podemos explorar o sistema de hierarquia de objetos.

Ciclo de vida de objetos Qt

- Sempre que criamos um novo objeto que herda a classe QObject podemos associar a ele um objeto pai através de seu construtor

```
#include <QObject>

class ItemGenerico : public QObject
{
    Q_OBJECT
public:
    explicit ItemGenerico(QObject *parent = nullptr);

signals:

};
```

Ciclo de vida de objetos Qt

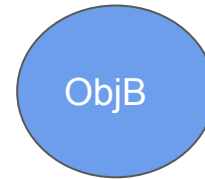
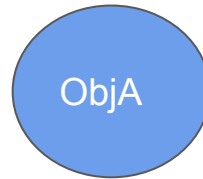
- Fazendo isso conseguimos realizar a destruição encadeada de objetos

Exemplo: considere a criação de 3 objetos:

```
#include "obja.h"
#include "objb.h"
#include "objc.h"

#include <QCoreApplication>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    ObjA *objA = new ObjA();
    ObjB *objB = new ObjB();
    ObjC *objC = new ObjC();
    return a.exec();
}
```

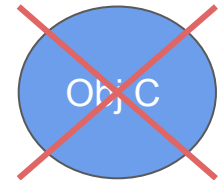
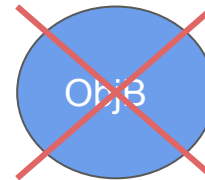
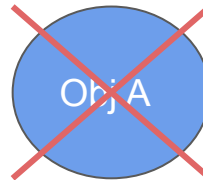


Ciclo de vida de objetos Qt

- Seguindo o padrão original do C++ todos os objetos precisam ser destruídos após a sua utilização, para que não aconteça um leak de memória

```
#include <QCoreApplication>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    ObjA *objA = new ObjA();
    ObjB *objB = new ObjB();
    ObjC *objC = new ObjC();
    delete objA;
    delete objB;
    delete objC;
    return a.exec();
}
```

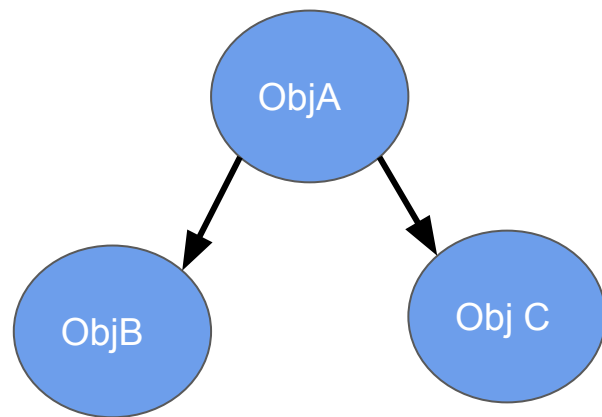


Ciclo de vida de objetos Qt

- No Qt podemos “ligar” objetos que herdam da classe QObject e integrar seus destrutores em cascata

```
#include <QCoreApplication>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    ObjA *objA = new ObjA();
    ObjB *objB = new ObjB(objA);
    ObjC *objC = new ObjC(objA);
    return a.exec();
}
```

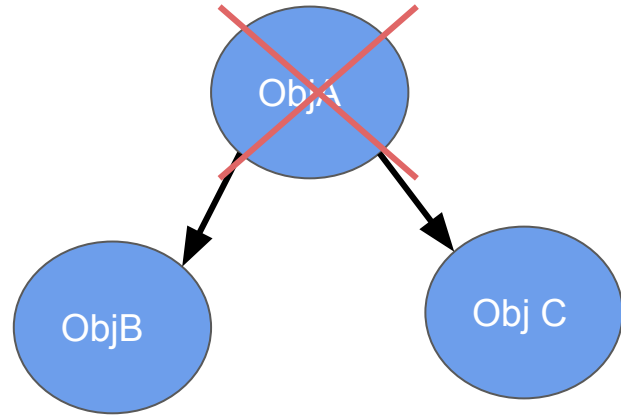


Ciclo de vida de objetos Qt

- No Qt podemos “ligar” objetos que herdam da classe QObject e integrar seus destrutores em cascata

```
#include <QCoreApplication>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    ObjA *objA = new ObjA();
    ObjB *objB = new ObjB(objA);
    ObjC *objC = new ObjC(objA);
    delete objA;
    return a.exec();
}
```

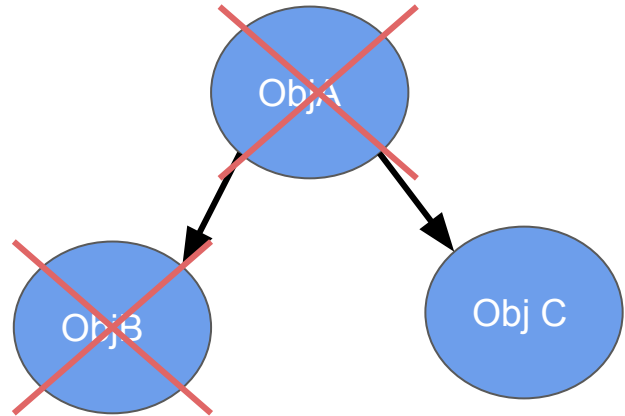


Ciclo de vida de objetos Qt

- No Qt podemos “ligar” objetos que herdam da classe QObject e integrar seus destrutores em cascata

```
#include <QCoreApplication>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    ObjA *objA = new ObjA();
    ObjB *objB = new ObjB(objA);
    ObjC *objC = new ObjC(objA);
    delete objA;
    return a.exec();
}
```

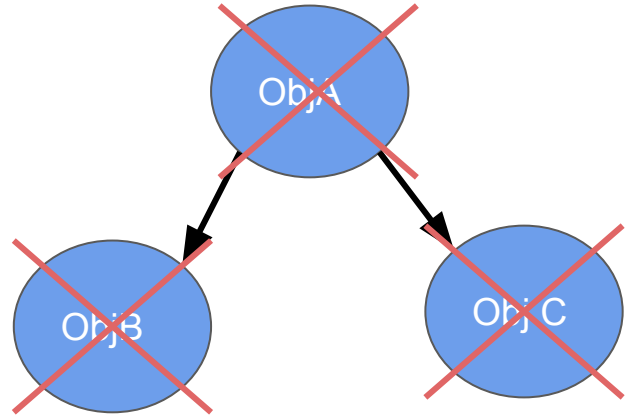


Ciclo de vida de objetos Qt

- No Qt podemos “ligar” objetos que herdam da classe QObject e integrar seus destrutores em cascata

```
#include <QCoreApplication>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    ObjA *objA = new ObjA();
    ObjB *objB = new ObjB(objA);
    ObjC *objC = new ObjC(objA);
    delete objA;
    return a.exec();
}
```



Ciclo de vida de objetos Qt

- Exemplo 2:

Abrir prática 05 - **CicloVidaObjeto**

Paralelizando a sua aplicação

- Como visto anteriormente, o Qt utiliza o sistema de Event Loop para gerenciar todo o fluxo de renderização, manipulação de eventos e execução de funções
- Porém, o que acontece se uma função consumir uma grande fatia de tempo ?



Paralelizando a sua aplicação

- Exemplo:

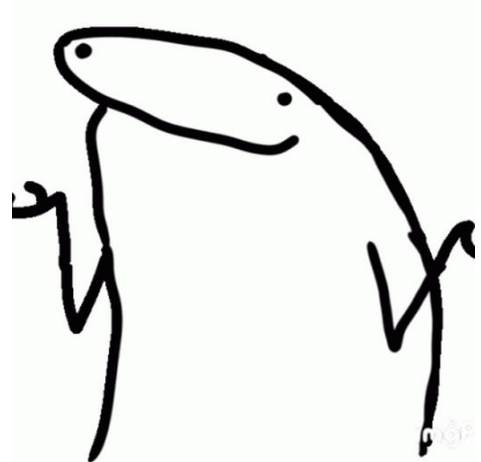
Abrir prática - **06-TarefaLongaSemThreads**

Testar Os seguintes valores de iteração

- 100
- 200
- 300
- 1000
- 4000
- 10000

Paralelizando a sua aplicação

- Por qual motivo a aplicação parou de responder ?



Paralelizando a sua aplicação

- Por padrão, o Qt utiliza uma única linha de execução (Thread) para controlar todos os eventos de tela, renderização e chamadas de função.

Paralelizando a sua aplicação

- Por padrão, o Qt utiliza uma única linha de execução (Thread) para controlar todos os eventos de tela, renderização e chamadas de função.
- Como resultado, a aplicação “trava” sempre que uma tarefa com alto custo computacional é implementada, já que ela toma conta da thread

Paralelizando a sua aplicação

- Por padrão, o Qt utiliza uma única linha de execução (Thread) para controlar todos os eventos de tela, renderização e chamadas de função.
- Como resultado, a aplicação “trava” sempre que uma tarefa com alto custo computacional é implementada, já que ela toma conta da thread
- Uma forma **NÃO RECOMENDADA** de mitigar esse problema é forçarmos uma execução do event loop dentro do método que está travando a thread principal

Paralelizando a sua aplicação

- Por padrão, o Qt utiliza uma única linha de execução (Thread) para controlar todos os eventos de tela, renderização e chamadas de função.
- Como resultado, a aplicação “trava” sempre que uma tarefa com alto custo computacional é implementada, já que ela toma conta da thread
- Uma forma **NÃO RECOMENDADA** de mitigar esse problema é forçarmos uma execução do event loop dentro do método que está travando a thread principal

```
QCoreApplication::processEvents()
```

Paralelizando a sua aplicação

- Por padrão, o Qt utiliza uma única linha de execução (Thread) para controlar todos os eventos de tela, renderização e chamadas de função.
- Como resultado, a aplicação “trava” sempre que uma tarefa com alto custo computacional é implementada, já que ela toma conta da thread.
- Podemos mitigar esse problema delegando essas tarefas para uma “thread trabalhadora” que, ao final da computação retornará para a thread principal o resultado final.

Paralelizando a sua aplicação

- Exemplo:
Abrir prática - **07-TarefaLongaComThreads**

Tipos de ponteiros em C++

- Manipular a memória de uma aplicação em tempo de execução é sempre uma grande dor de cabeça

Tipos de ponteiros em C++

- Manipular a memória de uma aplicação em tempo de execução é sempre uma grande dor de cabeça
- Principalmente se estamos falando de linguagens como C/C++

Tipos de ponteiros em C++

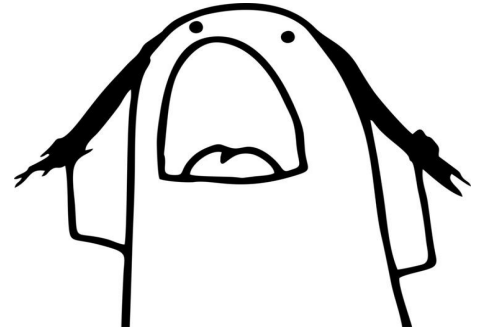
- Manipular a memória de uma aplicação em tempo de execução é sempre uma grande dor de cabeça
- Principalmente se estamos falando de linguagens como C/C++
- Uma vez que podemos alocar memória de várias formas diferentes em c++
 - a. Exemplos:

```
ObjetoDaora *objetoDaora = new ObjetoDaora();
```

```
ObjetoDaora *objetoDaora2 = (ObjetoDaora *)malloc(sizeof (ObjetoDaora));
```

```
ObjetoDaora *objetoDaora3 = new ObjetoDaora[100];
```

```
ObjetoDaora *objetoDaora4 = (ObjetoDaora *)malloc(sizeof(ObjetoDaora)*100);
```



Tipos de ponteiros em C++

- E como fica a destruição desses objetos ?
 - a. Em alocações de tipo simples, o desenvolvedor é responsável por destruir esses objetos
- Utilizando o operador **delete**

Tipos de ponteiros em C++

- Como podemos detectar um vazamento de memória em uma aplicação ?

Tipos de ponteiros em C++

- Como podemos detectar um vazamento de memória em uma aplicação ?
- O jeito mais simples de identificar um vazamento de memória é observar o comportamento do uso de memória durante seu uso
 - a. Criar e deletar objetos, e mesmo assim ver pouca, ou nenhuma redução no uso de memória

Tipos de ponteiros em C++

- Como podemos deletar um vazamento de memória em uma aplicação ?
- O jeito mais simples de identificar um vazamento de memória é observar o comportamento do uso de memória durante seu uso
 - a. Criar e deletar objetos, e mesmo assim ver pouca, ou nenhuma redução no uso de memória
 - b. Ferramentas de Benchmark de memória

Tipos de ponteiros em C++

- Como podemos deletar um vazamento de memória em uma aplicação ?
- O jeito mais simples de identificar um vazamento de memória é observar o comportamento do uso de memória durante seu uso
 - a. Criar e deletar objetos, e mesmo assim ver pouca, ou nenhuma redução no uso de memória
 - b. Ferramentas de Benchmark de memória
 - i. Valgrind (Linux)
 - ii. Heob (Windows)

Tipos de ponteiros em C++

- Como podemos deletar um vazamento de memória em uma aplicação ?
- O jeito mais simples de identificar um vazamento de memória é observar o comportamento do uso de memória durante seu uso
 - a. Criar e deletar objetos, e mesmo assim ver pouca, ou nenhuma redução no uso de memória
 - b. Ferramentas de Benchmark de memória
 - i. Valgrind (Linux)
 - ii. **Heob (Windows)**

Tipos de ponteiros em C++

- Felizmente no C++11, surgiram novos tipos de ponteiros com o objetivo de auxiliar o desenvolvedor no gerenciamento de memória
- Tipos de ponteiros inteligentes:
 - a. `unique_ptr`: é um tipo de ponteiro que não pode ser copiado e seu acesso é apenas existente dentro do escopo onde foi criado

Tipos de ponteiros em C++

- Felizmente no C++11, surgiram novos tipos de ponteiros com o objetivo de auxiliar o desenvolvedor no gerenciamento de memória
- Tipos de ponteiros inteligentes:
 - a. `unique_ptr`: é um tipo de ponteiro que não pode ser copiado e seu acesso é apenas existente dentro do escopo onde foi criado
 - b. `shared_ptr`: é um tipo de ponteiro que pode ser criado quando uma ou mais entidades de uma aplicação precisarem manipular um mesmo recurso

Tipos de ponteiros em C++

- Exemplo:
 - a. Imagine que você tenha uma imagem:



Imagem

Tipos de ponteiros em C++

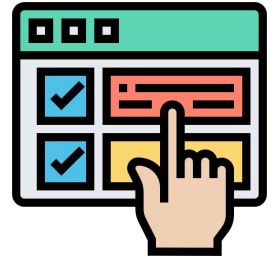
- Exemplo:
 - a. Imagine que você tenha uma imagem:
 - b. E dois componentes precisam mostrar essa imagem



Componente gráfico A



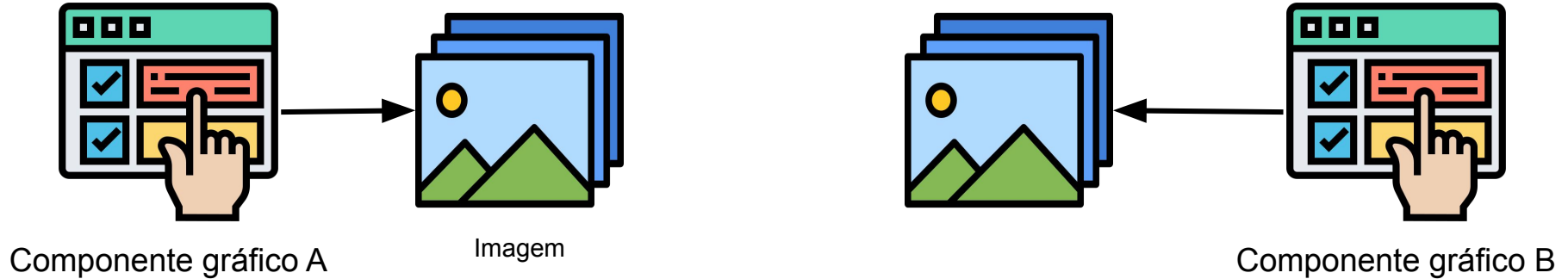
Imagem



Componente gráfico B

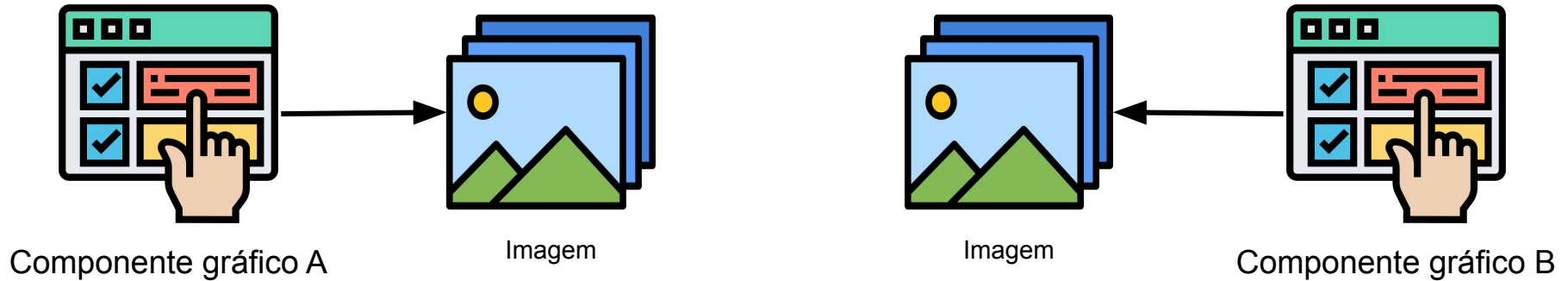
Tipos de ponteiros em C++

- Exemplo:
 - a. Podemos fazer isso de duas formas:
 - 1- Cada Componente tem a sua imagem



Tipos de ponteiros em C++

- Exemplo:
 - a. Vantagens:
 - i. Separação de responsabilidades
 - b. Desvantagens:
 - i. Uso de memória

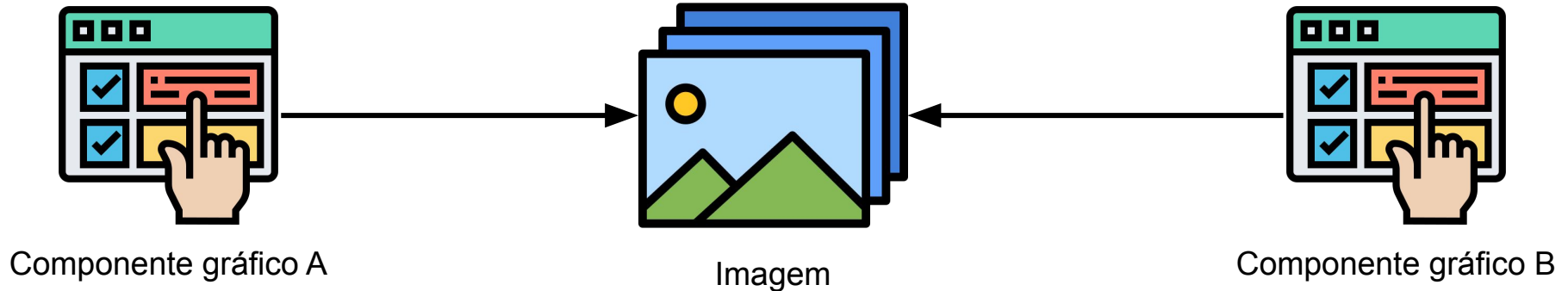


Tipos de ponteiros em C++

- Exemplo:

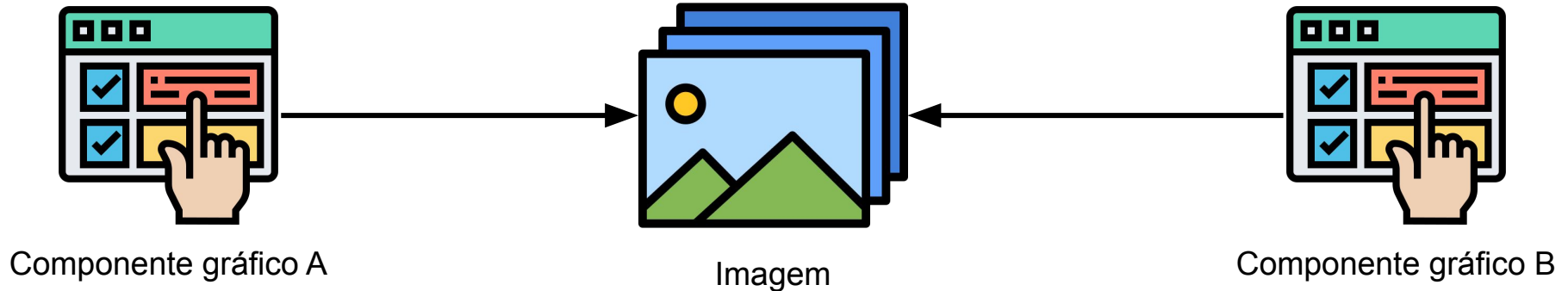
- a. Podemos fazer isso de duas formas:

2- Componentes Compartilhando a mesma imagem



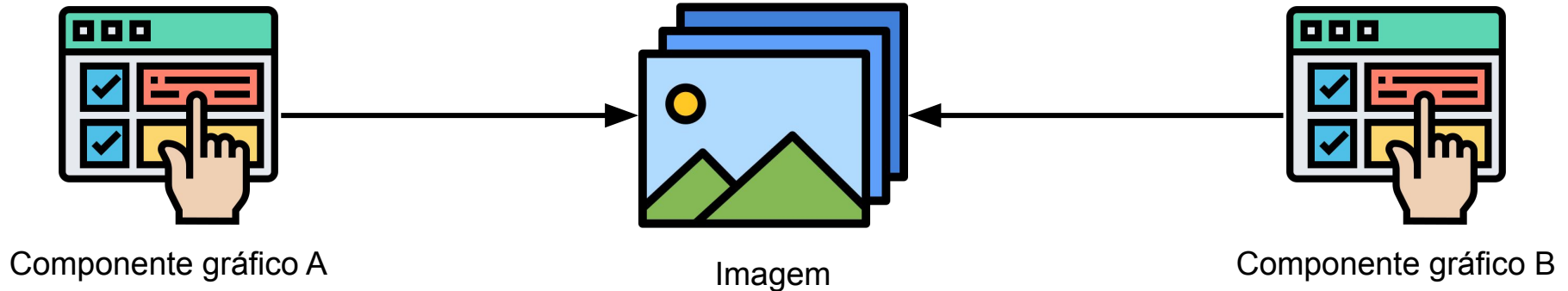
Tipos de ponteiros em C++

- Exemplo:
 - a. Vantagens:
 - i. Uso de memória
 - b. Desvantagens:
 - i. Confiabilidade no objeto



Tipos de ponteiros em C++

- Exemplo:
 - a. E se o componente gráfico A apagar a imagem ?
 - b. E se o componente gráfico B apagar a imagem ?



Tipos de ponteiros em C++

- Como estamos falando de memória tudo pode acontecer...



Tipos de ponteiros em C++

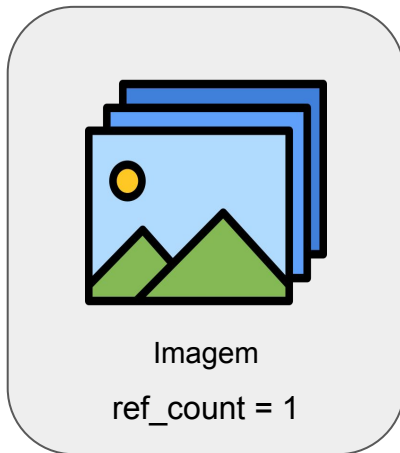
- Em um ponteiro de formato `std::shared_ptr`, conseguimos contornar esse problema de uma forma bem interessante



Imagem

Tipos de ponteiros em C++

- Em um ponteiro de formato `std::shared_ptr`, conseguimos contornar esse problema de uma forma bem interessante
- Quando encapsulamos nosso ponteiro original dentro de um ponteiro inteligente ele adquire uma característica chamada **contagem de referências**

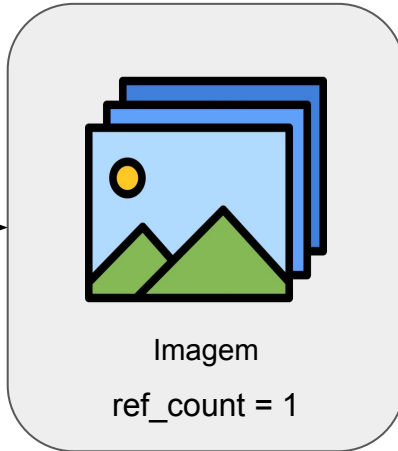


Tipos de ponteiros em C++

- Cada objeto que referencia um ponteiro inteligente incrementa o contador de referência

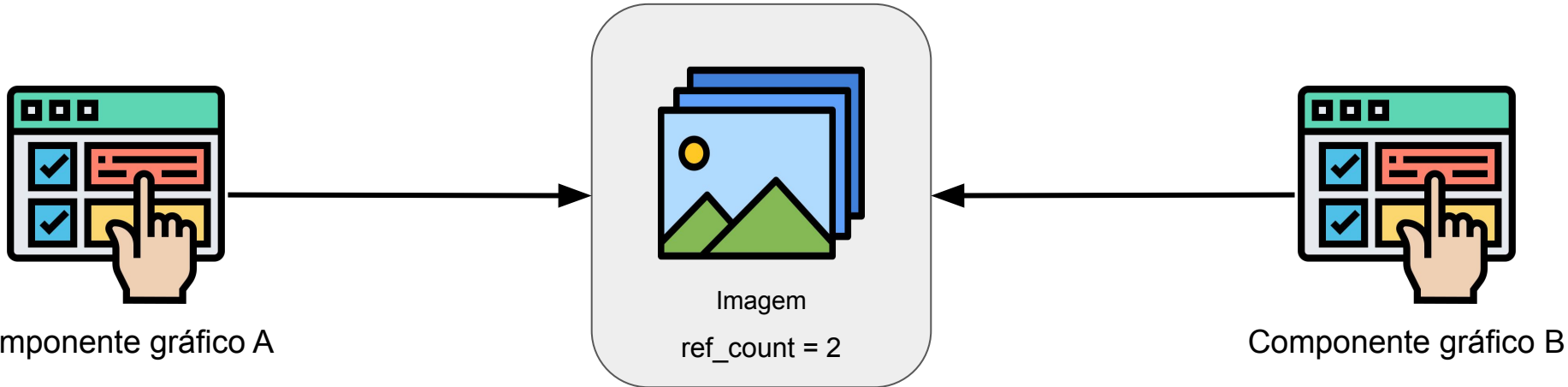


Componente gráfico A



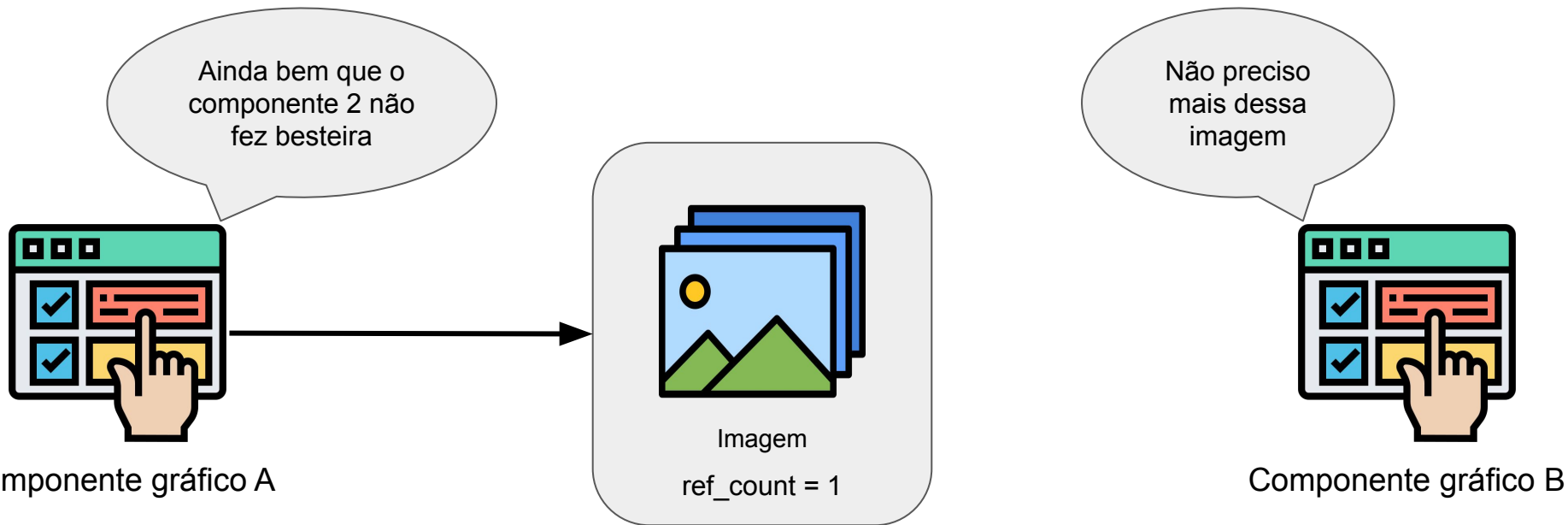
Tipos de ponteiros em C++

- Cada objeto que referencia um ponteiro inteligente incrementa o contador de referência desse ponteiro



Tipos de ponteiros em C++

- E sempre que um objeto deixa de referenciar esse ponteiro esse contador é decrementado

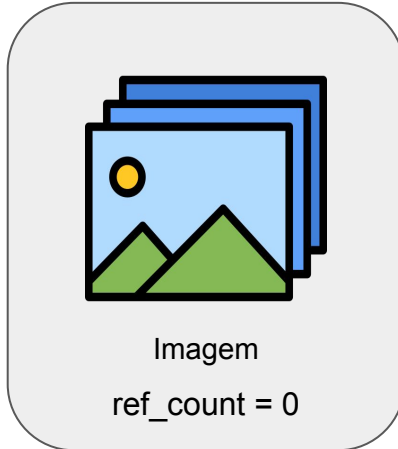


Tipos de ponteiros em C++

- Por fim, o objeto só é destruído quando seu contador de referências chega em 0



Componente gráfico A



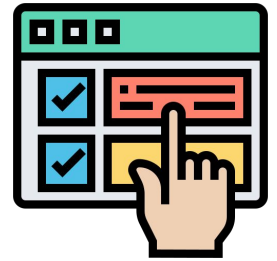
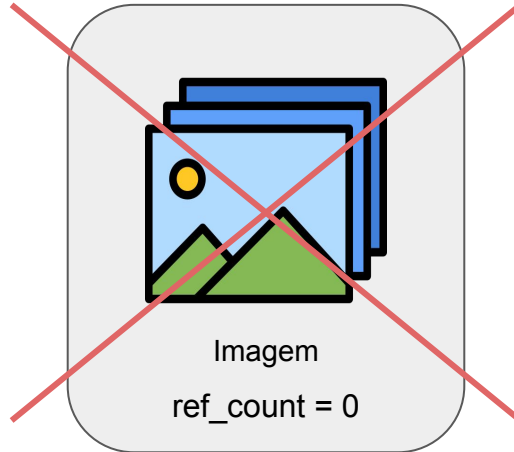
Componente gráfico B

Tipos de ponteiros em C++

- Por fim, o objeto só é destruído quando seu contador de referências chega em 0



Componente gráfico A



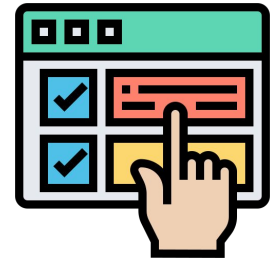
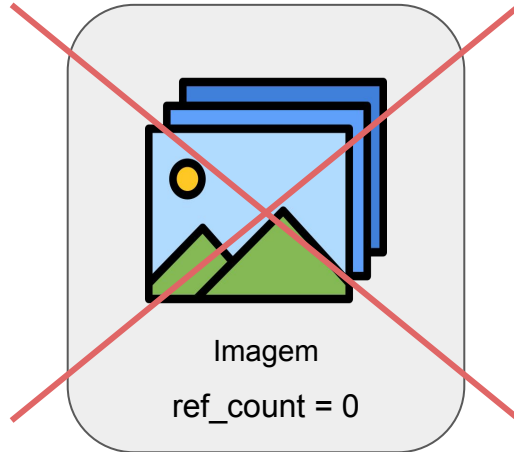
Componente gráfico B

Tipos de ponteiros em C++

- Por fim, o objeto só é destruído quando seu contador de referências chega em 0



Componente gráfico A



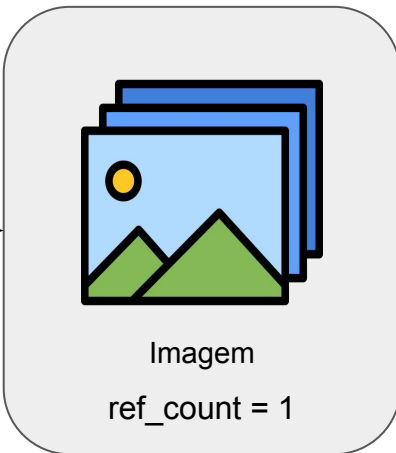
Componente gráfico B

Tipos de ponteiros em C++

- Só que se por algum motivo alguém não abandonar esse ponteiro, ele vai continuar existindo na memória



Componente gráfico A

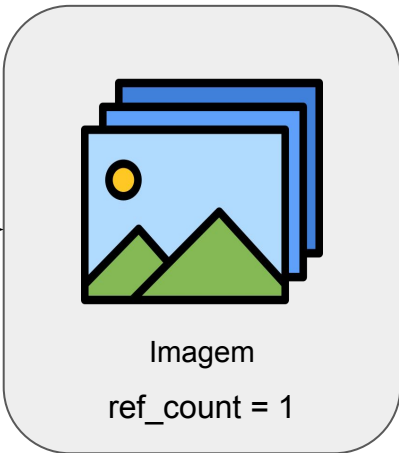


Tipos de ponteiros em C++

- Porém se esse ponteiro não perder todas as referências por algum motivo, ele continuará existindo na memória da aplicação até o final de sua execução
- Implicando em um vazamento de memória

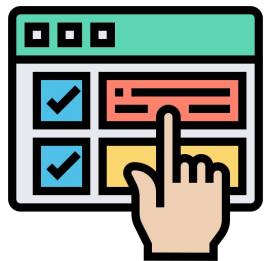


Componente gráfico A

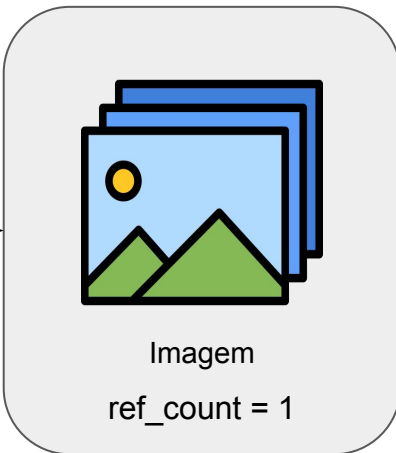


Tipos de ponteiros em C++

- Porém se esse ponteiro não perder todas as referências por algum motivo, ele continuará existindo na memória da aplicação até o final de sua execução
- Implicando em um vazamento de memória



Componente gráfico A



Tipos de ponteiros em C++

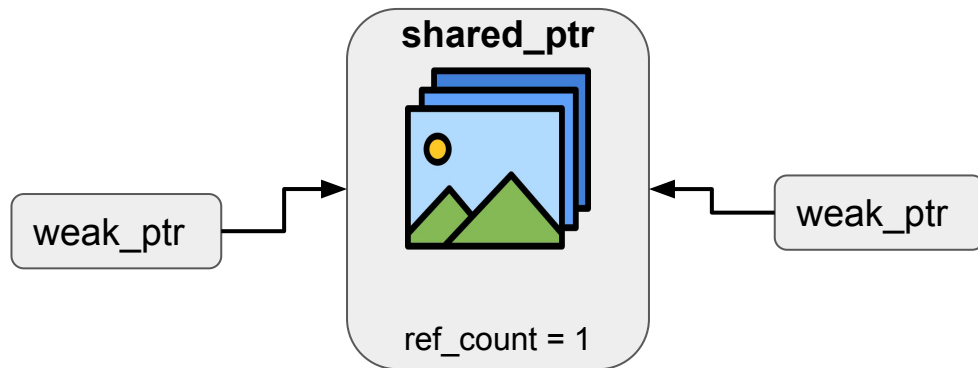
- Felizmente no C++11, surgiram novos tipos de ponteiros com o objetivo de auxiliar o desenvolvedor no gerenciamento de memória
- Tipos de ponteiros inteligentes:
 - a. `unique_ptr`: é um tipo de ponteiro que não pode ser copiado e seu acesso é apenas existente dentro do escopo onde foi criado
 - b. `shared_ptr`: é um tipo de ponteiro que pode ser criado quando uma ou mais entidades de uma aplicação precisarem
 - c. `weak_ptr`: é um tipo de ponteiro que pode ser utilizado combinado com o `shared_ptr`, para acesso de um objeto sem manipular seu contador de referências

Tipos de ponteiros em C++

- Ponteiros do tipo **weak_ptr** são capazes de compartilhar uma referência “fraca” para um objeto utilizando um **shared_ptr**

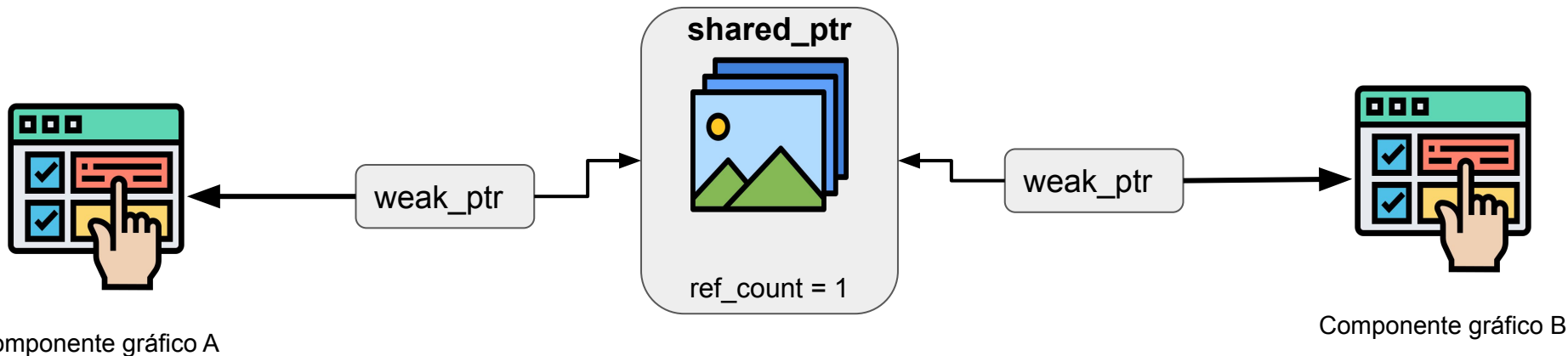
Tipos de ponteiros em C++

- Ponteiros do tipo **weak_ptr** são capazes de compartilhar uma referência “fraca” para um objeto utilizando um **shared_ptr**
- Vamos pegar o mesmo exemplo do **shared_ptr**, porém agora os objetos vão ter apenas acesso a uma referência fraca a esse ponteiro



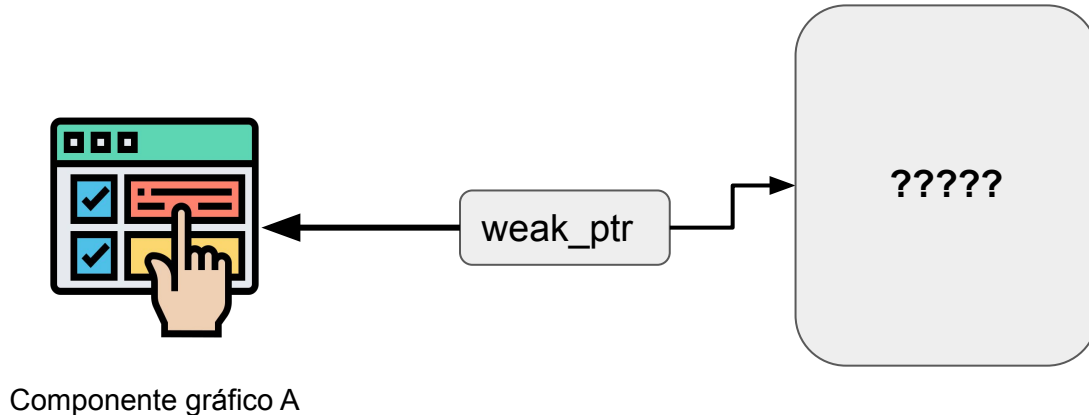
Tipos de ponteiros em C++

- O contador de referências ao objeto não sofre alteração, mesmo que os componentes acessem ou removam o acesso ao ponteiro, pois não é uma referência direta ao ponteiro



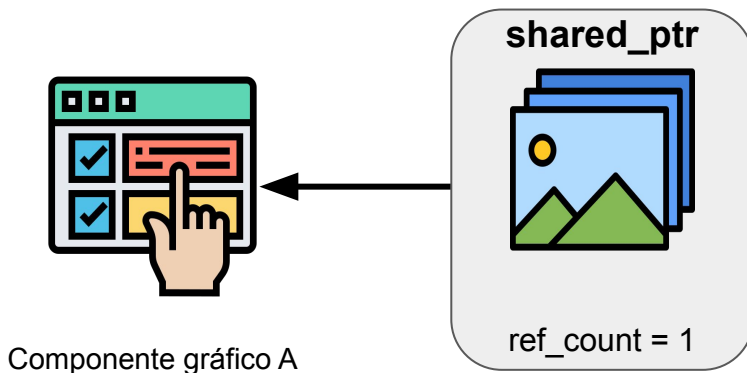
Tipos de ponteiros em C++

- Porém o ponteiro fraco em si, não é uma referência forte ao objeto (não conseguimos acessar os dados do ponteiro por ele diretamente



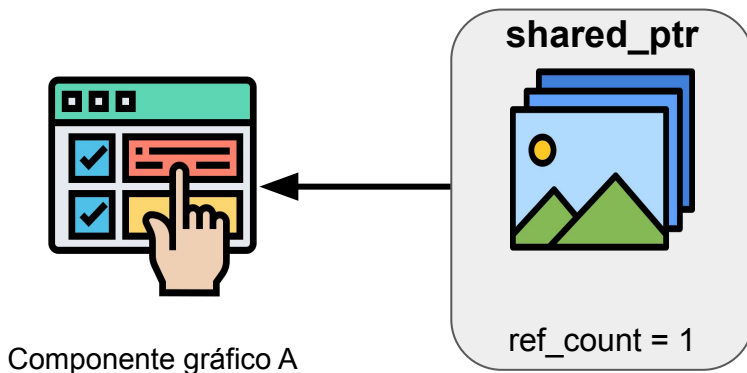
Tipos de ponteiros em C++

- Para isso precisamos fazer uma conversão do ponteiro fraco em um **shared_ptr**



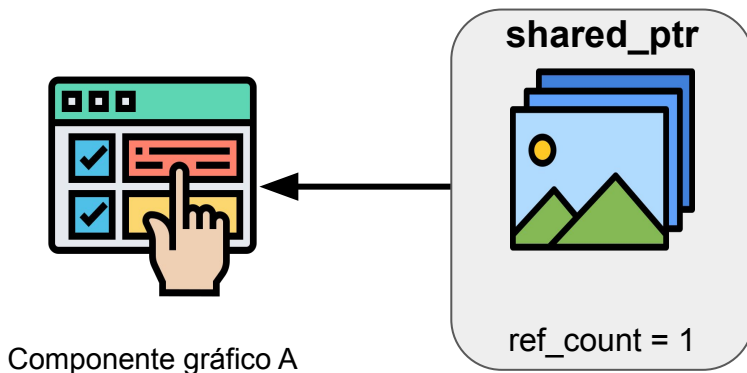
Tipos de ponteiros em C++

- Para isso precisamos fazer uma conversão do ponteiro fraco em um **shared_ptr**
- E é nesse momento que o componente é capaz de descobrir se o objeto ainda existe



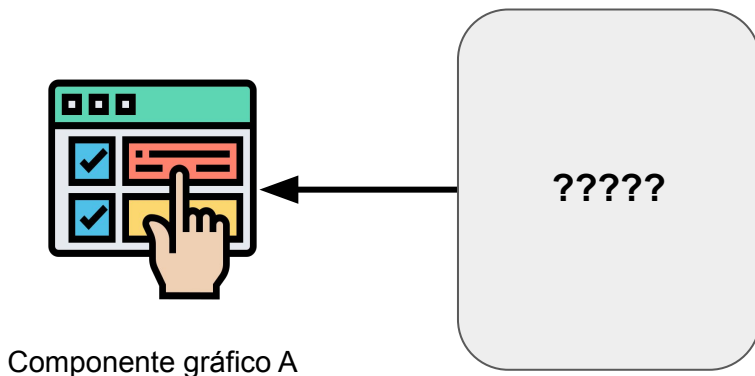
Tipos de ponteiros em C++

- Para isso precisamos fazer uma conversão do ponteiro fraco em um **shared_ptr**
- Se o **shared_ptr** ainda existir a conversão acontece corretamente



Tipos de ponteiros em C++

- Para isso precisamos fazer uma conversão do ponteiro fraco em um **shared_ptr**
- Caso contrário, um ponteiro inválido é criado e o componente pode identificar que aquela informação já não existe mais



Tipos de ponteiros em C++

- Abrir prática 08 - **Tipos de ponteiro**

That's all folks

- Meus contatos:
- Email: danilooalmeida94@gmail.com
- Instagram: dan.cpp (Posto algumas coisas de desenvolvimento no tempo livre :))

That's all folks

