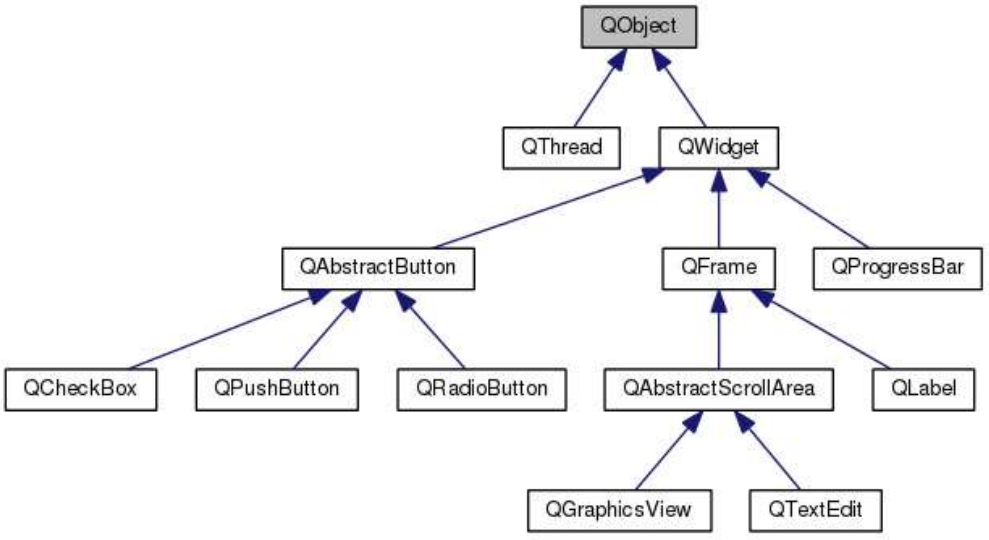
Qt

- Qt é principalmente uma biblioteca de programação em C++ cujo principal objetivo é possibilitar o desenvolvimento de interfaces gráficas e de aplicações multiplataforma (Linux, Windows, macOS, Android ou sistemas embarcados) a partir de um mesmo código fonte.
  - A pronúncia recomendada pelos criadores é "*kyut*", como a palavra em inglês "*cute*", fofo ou bonitinho. No Brasil, usa-se muito a pronúncia "*quê-tê*".
- Qt é conhecida principalmente pelo seu conjunto de ferramentas e de elementos gráficos de controle (chamados *widgets*) para a criação de interfaces de usuário gráficas (GUI), embora Qt também possa ser utilizada para o desenvolvimento de programas multiplataforma sem GUI, tais como ferramentas de comando de linha e programas para servidores.
- O Qt SDK é um *Software Development Kit* (SDK), ou seja, um conjunto de aplicações que permitem a criação de programas de computador usando Qt. No SDK são incluídos a biblioteca Qt e o ambiente de desenvolvimento integrado Qt Creator, além de outros aplicativos que operam em conjunto para permitir a geração e compilação de código fonte em C++ com essa biblioteca.
  - O Qt por si só não é um compilador, tratando-se apenas de uma biblioteca de programação. O compilador deve coexistir na máquina para que se possam produzir códigos executáveis.
- O ambiente Qt está disponível tanto sob licença de código aberto quanto comercial:
  - Qt código aberto: todos os usuários têm o direito de obter, modificar e redistribuir o código fonte completo do seu aplicativo.
  - Qt comercial: para desenvolvimento de aplicações de uso privativo.
- Instalação do Qt:
  - Certifique-se de ter uma conexão à rede rápida e sem restrição à quantidade de dados recebidos e ao menos 3Gb de espaço livre em disco: Qt é uma biblioteca grande!
  - Acesse a página do Qt (http://qt.io/) e escolha a opção Download, Open Source.
  - Leia as restrições para uso da versão de código aberto e escolha baixar o Qt Online Installer na versão adequada para seu sistema operacional.
  - Crie (é gratuito) ou acesse uma conta Qt para baixar o programa.
  - Especifique a pasta de instalação (C:\Qt por default).
  - Na escolha das versões e componentes Qt a instalar, não marque nada além das opções já pré-selecionadas, com exceção talvez do compilador MinGW caso não tenha instalado na sua máquina, pois pode aumentar MUITO a quantidade de dados a serem baixados.

Hierarquia de classes Qt

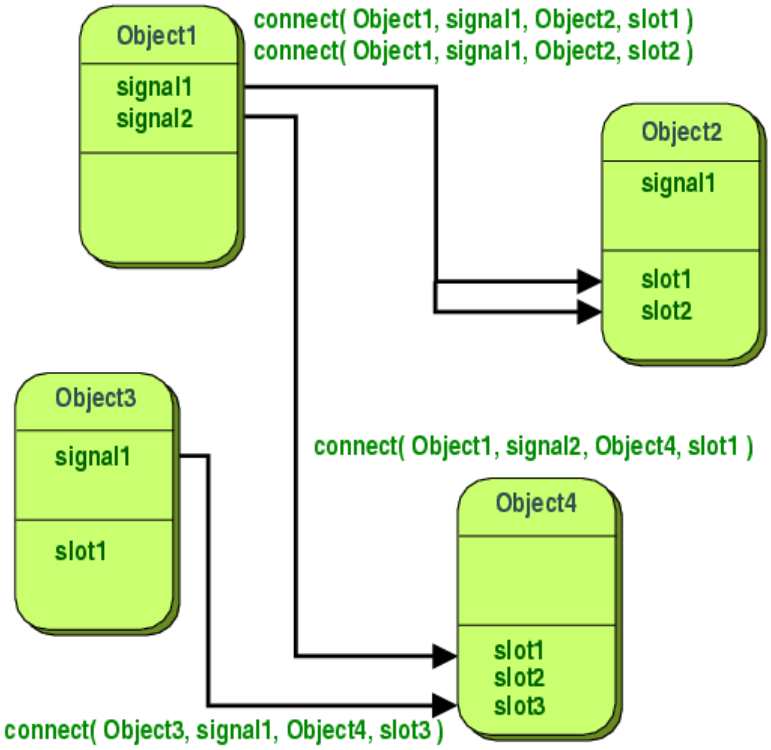


- Os widgets Qt são amplamente baseados em herança e métodos virtuais.
- `QObject` é a classe mais básica no Qt. A maioria das classes no Qt herda dessa classe. `QObject` fornece alguns recursos essenciais:
  - Nome do objeto: você pode definir um nome (uma string) para um objeto.
  - Sistema parental: qualquer objeto que herda de `QObject` pode ter um pai e filhos.
  - Sinais e slots: mecanismo de comunicação entre objetos.
- O widget mais básico é o `QWidget`, que herda do `QObject` e do qual todos os outros widgets herdam. O `QWidget` contém a maioria das propriedades usadas para descrever uma janela ou um widget, como posição e tamanho, cursor do mouse, tooltips, etc.
- O sistema parental entre objetos torna alguns comportamentos mais simples de gerenciar:
  - Quando um objeto é destruído, todos os seus filhos também são destruídos. Portanto, chamar `delete` se torna opcional na maioria dos casos.
  - Widgets filhos aparecem automaticamente dentro do widget pai.
- Ao criar uma classe que herda de um widget Qt, para garantir que as funcionalidades da classe `QObject` funcionem nos objetos da sua classe, você deve incluir a macro `Q_OBJECT` logo no início da declaração da classe e prever como parâmetro do construtor default um ponteiro para o objeto pai do novo objeto a ser criado:

```
class MyWidget : public QWidget
{
  Q_OBJECT

  public:
    explicit MyWidget(QWidget *parent = 0);
  …
}
```

Sinais e slots



- Na programação de interfaces gráficas (GUI), quando alteramos um widget, geralmente queremos que outro widget seja notificado. Por exemplo, se um usuário clicar no botão Fechar, provavelmente queremos que a função close () da janela seja chamada.
- Em Qt, sinais e slots são usados para comunicação entre objetos.
- Sinais são emitidos por objetos quando mudam de estado de uma maneira que pode ser interessante para outros objetos ou quando a função `emit` é chamada. Os widgets têm muitos sinais predefinidos, mas pode-se criar classes que herdam dos widgets para adicionar novos sinais.
- Um slot é uma função que é chamada em resposta a um sinal específico. Os widgets têm muitos slots predefinidos, mas é uma prática comum criar classes que herdam dos widgets e adicionar novos slots para lidar com sinais específicos.
- O objeto não sabe nem se importa se alguma coisa está recebendo os sinais que emite. A emissão de um sinal por um objeto só acarretará a execução de um determinado slot em outro objeto se eles estiverem conectados através da função `connect`.
- É possível conectar quantos sinais se deseja em um único slot e um sinal pode ser conectado a quantos slots forem necessários.
- A assinatura (número e tipo de parâmetros) de um sinal deve corresponder exatamente à assinatura do slot ao qual ele seja conectado.
- Os slots podem ser usados para receber sinais, mas também são funções membro normais que podem ser chamadas explicitamente no código do programa. No entanto, como slots, eles podem ser chamados por qualquer componente, independentemente do seu nível de acesso, através de uma conexão sinal-slot. Isso significa que um sinal emitido de uma instância de uma classe arbitrária pode fazer com que um slot seja “executado” mesmo sendo privado.
- Sinais são declarados com uma sintaxe similar a funções, mas não devem ser implementados pelo programador em um arquivo `.cpp`. As “funções” dos sinais sempre retornam `void`.

Métodos frequentemente utilizados

Geral:

- Exibir/esconder um widget:
  - show()/hide()
- Habilitar/desabilitar sensibilidade de um widget a eventos
  - setEnabled(true ou false)
- Acesso às propriedades de um widget:
  - set<Nome>(...) – altera o valor da propriedade <Nome>
  - <Nome>() – retorna o valor atual da propriedade <Nome>

Janela principal do programa:

- Definir o título:
  - Alterar propriedade windowTitle
- Definir o ícone da aplicação:
  - QPixmap Img;
    setWindowIcon(QIcon(Img));
- Encerrar a execução:
  - QCoreApplication::quit().

QString:

- Substituindo valores dentro de uma QString (similar a printf):
  - QString(“Texto %1 texto %s texto”).arg1(...).arg2(...)
    Substitui as expressões %1 %2 etc. pelos parâmetros de arg1() arg2() etc.
- Gerando a string que corresponde a um número:

  - QString prov = QString::number(...)
- Convertendo para strings C++ e C:
  - QString prov;
    ... prov. toStdString() ... // string C++
    ... prov. toLatin1().data() ... // const char*

QLabel:
- Alinhamento do texto:
  - setAlignment(<ALIGN_H> | <ALIGN_V>)
    ALIGN_H = Qt::AlignLeft OU Qt::AlignHCenter OU Qt::AlignRight
    ALIGN_V = Qt::AlignTop OU Qt::AlignVCenter OU Qt::AlignBottom
    Qt::AlignCenter = Qt::AlignHCenter | Qt::AlignVCenter
- Dimensão:
  - setGeometry(0,0,30,30)
- Cor de fundo:
  - setStyleSheet("background: ...")
- Alterar a informação:
  - setNum(valor) – exibe número
  - setText("...") – exibe texto
  - setPixmap(...) – exibe imagem

QSpinBox:
- Faixa de valores de 0 a 9:
  - setRange(0,9)
- Comportamento circular (8→9→0→1):
  - setWrapping(true)
- Valor mínimo com significado especial, devendo ser exibido texto ao invés do valor:
  - setSpecialValueText("...")
- Fixar / recuperar o valor atual:
  - ui->... ->setValue(...) / ui->... ->value()

QComboBox:
- Fixar as opções:
  - ui->...->addItems(QStringList() << "..." << ... << "...")
- Fixar / recuperar a opção atual:
  - ui->...-> setCurrentText("...") / ui->...->currentText()

QLineEdit
- Fixar / recuperar o conteúdo atual:
  - setText("...") / text()
- Limpar o conteúdo atual:
  - ui->...->clear();
- Modo de entrada que não exibe os caracteres digitados (para ler senhas):
  - ui->...->setEchoMode(QLineEdit::Password);

Imagens:
- Dois tipos de widgets para armazenar imagens:
  - QImage: otimizada para entrada/saída de dados e para acesso e manipulação direta dos pixels. Permite leitura e salvamento de arquivos nos formatos populares (JPEG, PNG, etc)

  - QPixmap: otimizada para exibir imagens em tela. Só pode ser manipulada através de QPainter. Pode ser convertida para uma QImage.
- Criando imagem com tamanho inicial:
  - QPixmap Img(40, 40);
- Preenchendo com uma cor:
  - QColor cor(R,G,B);
    Img.fill(cor);
- Desenhando:
  - QPen pen;
    QPainter painter;
    pen.setWidth(3);
    pen.setColor(Qt::red);
    painter.begin(&Img);
    painter.setPen(pen);
    painter.drawLine(10,29, 29,10);
    painter.drawEllipse(QPoint(20,20), 12, 12);
    painter.end();
- Lendo imagem de arquivo:
  - QPixmap Img;
    Img.load(<nome arquivo>,<tipo>);

QStatusBar:
- Já existe um objeto do tipo QStatusBar dentro de um objeto do tipo QMainWindow:
  - Acessível através de ui->(nome do objeto)-> ... ou de statusBar()-> ...
- Exibir/apagar mensagens:
  - showMessage e clearMessage
- Exibir um widget:
  - ui->statusBar->insertWidget(0,new QLabel("..."));
    QLabel *prov = ...;
    ui->statusBar->insertWidget(1,prov);

QTableWidgetItem:
- Texto centralizado:
  - setTextAlignment(Qt::AlignCenter)
- Dimensão:
  - setGeometry(0,0,30,30)
- Cor de fundo:
  - setStyleSheet("background: ...")
- Alterar a informação:
  - setNum(valor) – exibe número
  - setText("...") – exibe texto
  - setPixmap(...) – exibe imagem
- Texto em negrito:
  - QFont ff;
    ff.setBold(true);
    … ->setFont(ff);

QTableWidget:
- Atualizar (redesenhar) depois que o conteúdo de algum item foi mudado:
  ui-> ... ->viewport()->update()
- Fixar/redimensionar o número de linhas e colunas:

  - setRowCount / setColumnCount
- Exibir cabeçalhos:
  - propriedades horizontalHeaderVisible e verticalHeaderVisible
- Dimensões das células:
  - propriedades verticalHeaderDefaultSectionSize, verticalHeaderMinimumSectionSize, horizontalHeaderDefaultSectionSize e horizontalHeaderMinimumSectionSize
- Desabilitar seleção de células:
  - setSelectionMode(QAbstractItemView::NoSelection)
- Desabilitar seleção múltipla de células (shift-click, cntrl-click):
  - setSelectionMode(QAbstractItemView::SingleSelection)
- Selecionar linha inteira ao clicar em célula:
  - setSelectionBehavior(QAbstractItemView::SelectRows)
- Desabilitar navegação usando TAB:
  - setTabKeyNavigation(false)
- Exibir grade:
  - setShowGrid(true ou false)
- Definindo os cabeçalhos:
  - ui->...->setHorizontalHeaderLabels(QStringList() << "..." << ... << "...")
  - QTableWidgetItem *prov = new QTableWidgetItem("...");
    ... (altera as propriedades de prov)
    ui->... ->setHorizontalHeaderItem(<num coluna>, prov);
- Definindo a cor de fundo dos cabeçalhos:
  - ui->... ->setStyleSheet("QHeaderView::section { background-color:lightgray }")
- Definindo se uma coluna inteira pode ser selecionada ao clicar no cabeçalho:
  - ui->... ->horizontalHeader()->setSectionsClickable(false)
- Mesclar / separar células:
  - ui->...->setSpan(...)
  - ui->...->clearSpans()
- Determinando como as colunas devem se expandir quanto a tabela mudar de tamanho:
  - ui->...->horizontalHeader()->setSectionResizeMode(0,QHeaderView::ResizeToContents)
    ui->...->horizontalHeader()->setSectionResizeMode(1,QHeaderView::Stretch)
- Inserir um conteúdo em uma célula da tabela:
  - QLabel *prov = new QLabel;
    ... (altera as propriedades de prov)
    ui->... ->setCellWidget(i,j,prov)
  - QTableWidgetItem *item= new QTableWidgetItem;
    ... (altera as propriedades de item)
    ui->... ->setItem(i,j,item)
- Alterar o conteudo de uma célula da tabela:
  - QLabel *prov = ui->... ->cellWidget(i,j);
    ... (altera as propriedades de prov)
  - QTableWidgetItem *item = ui->... ->item(i,j);
    ... (altera as propriedades de item)
- Apagar:
  - ui->...->clear() – apaga tudo.
  - ui->...->clearContents() – apaga o conteúdo, deixa os cabeçalhos
- Selecionar uma célula/faixa de céluas:
  - ui->...->setCurrentCell() – seleciona célula.
  - ui->...->setRangeSelected() – seleciona faixa de células.

Slots que respondem a eventos:

- Conexão de um sinal a um slot:
  - connect(&<widget_sinal>, SIGNAL(<nome_sinal>), &<widget_slot>, SLOT(<nome_slot>)
- Para criar automaticamente um slot já conectado a um sinal específico, clique com o botão direito do mouse no widget ou no nome da ação (em baixo do designer) e escolha “Go to slot”; em seguida, escolha o nome do sinal com o qual você quer associar um slot.
  - on_..._cellClicked(int row, int column) – clicar em uma célula de uma QTableWidget
  - on_..._valueChanged(int arg1): alterar o valor de um QSpinBox
  - on_..._clicked(): pressionar um QPushButton
  - on_action..._triggered(): escolher uma ação de um menu

Leitura e salvamento de arquivos:

- Funções que retornam o nome do arquivo escolhido:
  - QString fileName = QFileDialog::getOpenFileName(this, [título], [pasta], [tipos]);
  - QString fileName = QFileDialog::getSaveFileName(this, [título], [pasta], [tipos]);
- Associar um arquivo a um nome:
  - QFile file(fileName)
- Abrir um arquivo para leitura:
  - file.open(QIODevice::ReadOnly) – retorna true em caso de sucesso
- Ler todo o conteúdo de um arquivo aberto para leitura:
  - QTextStream in(&file);
    Conteúdo disponível na QString: in.readAll()
- Abrir um arquivo para escrita:
  - file.open(QIODevice::WriteOnly) – retorna true em caso de sucesso
- Escrever em um arquivo aberto para escrita:
  QTextStream out(&file);
  out << ... ;
  out.flush();
- Fechar um arquivo:
  - file.close()

Caixas de diálogo:

- Mensagem em geral:
  - QMessageBox msgBox;
    msgBox.setText("...”);
    msgBox.exec();
- Erros:
  - QMessageBox::critical(this, [Título], [Mensagem]);
  - QMessageBox::warning(this, [Título], [Mensagem]);

Espera por um evento:

- Cria-se um loop de espera por um evento, associando-se o sinal que se quer esperar com o slot de terminar o loop:
  - QEventLoop espera;
    connect(ui->botaoContinuar, SIGNAL(clicked()), &espera, SLOT(quit()));
    espera.exec();