1. Этапы построения проблемно-центрированного дизайна.

анализ задач и пользователей;

•

выбор репрезентативных задач;

•

заимствование;

•

черновое описание дизайна;

•

обдумывание дизайна;

•

прототипа;

создание макета или

•

пользователями;

тестирование дизайна с

•

итерирование;

•

реализация;

•

отслеживание эксплуатации;

•

изменение дизайна.

1. CWT анализ.

Пример – CD копировщик. Cognitive walkthrough. Формализованный способ представления мыслей и действий людей, когда они пользуются интерфейсом в первый раз. \*Все алгоритмы действий пользователя, как он их себе представляет должны давать результат\*. Идём по действиям алгоритма и помечаем как нужно улучшить интерфейс, чтобы \* этот пункт выполнялся. Дорлжна быть обратная связь от программы. Для CWT нужно сразу иметь список действий для решения задачи. Если пользователь не знает, что-то ещё д.б. сделано, то пусть программа это сделает сама.

1. GOMS анализ.

Goals Operations Methods Selections. Подсчёт времени выполнения опытным пользователем. Время затрачиваемое машиной на выполнение операций без участия человека не должно учитываться. Семантические единицы: К 0.2 с – нажатие клавиши, М 1.35 с – мыслительная деятельность, В 0.2 с – клик мышью, Р 1.1 с – point – указание мышью, R – ожидание ответной реакции компьютера, H 0.4 с – hover перенос руки с клавиатуры на мышь или наоборот, D - проведение с помощью мыши прямой линии (например, выделение или прокрутка текста);

Результат – оценка среднего времени , понимание того, какие улучшения можно провести.

1. Правила Нильсена-Молиха.
2. Простой и естественный диалог – информация только по теме
3. Говорите на языке пользователя
4. Минимизируйте загрузку памяти пользователя(введенная информация должна оставаться на экране как можно дольше.
5. Будьте последовательны(должна быть возможность изучить действия в одной части программы и применить их снова).
6. Обеспечьте обратную связь
7. Обеспечьте хорошо обозначенные выходы(из окон доп. Свойств например).
8. Обеспечьте быстрые клавиши и ярлыки
9. Хорошие(информативные) сообщения об ошибках
10. Предотвращайте ошибки(например при переключении калькулятора в двоичный режим можно отключить цифры 2-9).
11. Принципы организации графического интерфейса.

1)Принцип кластеризации(напр. кнопки для отдельных функций располагаются рядом в блоках с названиями)

2) «Видимость отражает полезность» - наиболее часто используемые элементы на самом видном месте

3)Принцип интеллнектуальной последовательности(экраны с похожими функциями д.б. похожи, напр. Кнопка «помощь» д.б. всегда в одном и том же месте

4) цвет в последнюю очередь (цвет не однозначен для разных культур

5) Принцип уменьшения беспорядка(одинаковые шрифты).

1. Общий алгоритм работы со списком QList.

Нежелательно использовать [ ], вместо этого лучше использовать at( )

QList<int> list;//объявили

list << 10 << 20 << 30;//внесли данные

Для перечисления элементов:

QValueList<int>::iterator it = list.begin();

while (it != list.end()) {

qDebug() << "Element:" << \*it;

++it;

}

move Перемещает элемент с одной позиции на другую

rernoveFirst Удаляет первый элемент списка

rernoveLast Удаляет последний элемент списка

-

swap Меняет местами два элемента на указанных позициях

---- --

takeAt Возвращает элемент на указанной позиции и удаляет его

~--

takeFirst Удаляет первый элемент и возвращает его

~----- -

takeLast Удаляет последний элемент и возвращает его

-- --------

toSet Возвращает контейнер QSet<T> с данными, содержащимися в объекте

QList<T>

~ --- - ----------

toStdList Возвращает стандартный список STL std:: list<T> с элементами,

содержащимися в объекте QList<T>

toVector Возвращает объект вектора QVector<T> с элементами, содержащимися в объекте QList<T>

1. Общий алгоритм работы с текстовыми файлами в Qt.

QFile file(“file.txt”);//Создали объект типа QFile.

if (file.open(QIODevice::WriteOnly))// Если записываемый файл не существует, он будет создан. В противном случае он будет перезаписан.

{

    file.write("Test string\n");

    file.write("Test string2");

    file.close();

}

if ((file.exists())&&(file.open(QIODevice::ReadOnly)))//построчное чтение

{

    QString str="";

    while(!file.atEnd())

    {

        str=str+file.readLine();

    }

    ui->textBrowser->setText(str);

    file.close();

}

if ((file.exists())&&(file.open(QIODevice::ReadOnly)))//считали сразу всё

{

    ui->textBrowser->setText(file.readAll());

    file.close();

}

*if*(file.open(QIODevice::Append)){ *//will* *write* *to* *the* *end*

QTextStream stream (&file);//создал поток, ввёл в него ссылку на QFile

QString str = "\na new line";//Создал строчку

stream<<str.toUpper();//ввёл в поток строчку, да ещё и прописными бёквами

*if*(stream.status()!=QTextStream::*Ok*){

qDebug()<<"Ошибка чтения файла";

}

file.close();

}

1. Механизм работы сигналов и слотов.

Создан для обеспечения ООП. СиС – средства, позволяющие эффективно производить обмен информацией о событиях, производимых объектами. Q\_OBJECT – необходимый макрос. Объект "телефон" выслал сигнал "звонок", на который объект "человек" отреагировал слотом "снятия трубки". Не явл. Частью С++. Сигнал объявлен в одном месте, функция им запускаемая в другом. Можно сделать pudlic void sendSignal(){emit doIt();} Т.о. сигнал doIt(); м.б. вызван другим классом. Слоты – методы, которые присоединяются к сигналам. Связь может производиться между разными потоками(объектами).

1. Менеджер компоновки: определение, виды, назначение.

Позволяет размещать виджеты/компоновки на поверхностях других виджетов или компоновок.

QHBoxLayout//это суётся в виджет, а может и в другой Layout быть засунуто

QVBoxLayout

QGridBoxLayout

QBoxLayout принимает параметры( leftToRight и т.п. для указания мест размещения)

# include <QtWidgets>

int main(int argc, c har\*\* argv )

{QApplication app(argc, argv);

QWidget wgt;

QPushButton\* pcmdA new QPushButton ("А");

QPushButton\* pcmdB new QPushButton("B");

QPushButton\* pcmdC new QPushButton 1 "С") ;

QPushButton\* pcmdD new QPushBut ton 1 "D") ;

QGridLayout\* pgrdLayout = new QGridLayout;//создали компоновку табличного размещения

pgrdLayout->setContentsMargins(5, 5, 5, 5);//отступ от границы

pgrdLayout->setSpacing(l5);//расст. Между виджетами

pgrdLayout->addWidget(pcmdA, О, 0);

pgrdLayout->addWidget(pcmdB, о, 1);

pgrdLayout->addWidget(pcmdC, 1, 0);

pgrdLayout->addWidget(pcmdD, 1, 1) ;

wgt.setLayout(pgrdLayout);

wgt.show();

return арр.ехес();

}

1. Стандартные диалоговые окна работы с файлами в Qt.

QString str = QFileDialog::getOpenFileName(0, "Open Dialog", "", "\*.cpp \*.h");

// str will contain a full path to a file

Потом можно сделать например QFile file(str);

getSaveFileName// вызывает окно сохранения файла

getExistingDirectory// выбрать папку. Также возвращает путь до выбранного каталога.

1. Стандартное диалоговое окно настройки принтера в Qt.

QT += printsupport прописываем в .pro

QPrinter\* printer;

QPrintDialog \* pPrintDialog = new QPrintDialog(printer);

If(pPrintDialog ->exec() == QDialog::Accepted)

{draw(printer);//можно так

// ну или печатаем выбранный документ

}

void Printer::draw(QPaintDevice\* ppd)

{

QPainter painter(ppd);

QRect r(painter.viewport());

painter.setBrush(Qt::*white*);

painter.drawRect(r);

painter.drawLine(0, 0, r.width(), r.height());

painter.drawLine(r.width(), 0, 0, r.height());

painter.setBrush(Qt::*NoBrush*);

painter.setPen(QPen(Qt::*red*, 3, Qt::*DashLine*));

painter.drawEllipse(r);

painter.setPen(Qt::*blue*);

painter.setFont(QFont("Times", 20, QFont::*Normal*));

painter.drawText(r, Qt::*AlignCenter*, "Printer Test");

} delete pPrintDialog;

1. Стандартное диалоговое окно выбора цвета, выбора шрифта в Qt.

QColor color = QColorDialog::getcolor(blue);

If(!color.isValid()){//cancel

}

//isValid возвр true, если была нажата Ok

Bool bOk;

QFont fnt = QFontDialog::getFont(&bOk);

If(!bOk){/\*cancel was pressed\*/}

1. Стандартное диалоговое окно ввода, окно прогресса в Qt.

Bool bOk;

QString str = QInputDialog::getText(0, "Input", "Label:", QLineEdit::Normal, “smth already written”, &bOk);

If(!bOk){/\*cancel was pressed\*/}

int n = 100000;

QProgressDialog\* pprd = *new* QProgressDialog("Processing the data...", "&Cancel", 0, n);

pprd->setMinimumDuration(0);*//задержка* *показа* *диалогового* *окна.* *Как* *она* *работает* *хз.* задержка показа окна, Или типа если ожидается потраттить(на прогресс) меньше этого времени, то окно не показывать. Наверное так

pprd->setWindowTitle("Please Wait");

*for* (int i = 0;i<n;++i) {

pprd->setValue(i);

qApp->processEvents();

*if*(pprd->wasCanceled()){

*break*;

}

}

pprd->setValue(n);

*delete* pprd;//.

1. Три типа окон сообщений в Qt.

QMessageBox::information(0, "Information", "Operation Complete");*//informational*

*////a* *warning* *message* *with* *options:* *yes* *or* *no*

int n = QMessageBox::warning(0, "Warning", "The text in the file has changed"

"\n Do you want to save the changes?",

QMessageBox::*Yes* | QMessageBox::*No*,

QMessageBox::*Yes*);

*if*(n==QMessageBox::*Yes*){

*//Saving* *the* *changes*

}

(*new* QErrorMessage(*this*))->showMessage("Write Error");*//obviously* *error* *message*

1. Воздушные подсказки, подсказка «Что это?» в Qt.

QPushButton\* pcmd = *new* QPushButton("&Ok");*//tooltip(всплывающая* *подсказка)*

pcmd->setToolTip("Button");

pcmd->show();

#include <QtWidgets>

*class* MyWidget : *public* QWidget {

*private*:

QLabel\* m\_plblToolTip;

*protected*:

*virtual* bool *event*(QEvent\* pe)

{

*if* (pe->type() == QEvent::*ToolTip*) {

QHelpEvent\* peHelp = *static\_cast*<QHelpEvent\*>(pe);

m\_plblToolTip->move(peHelp->globalPos());

m\_plblToolTip->setText(toolTip());

m\_plblToolTip->show();

QTimer::singleShot(3000, m\_plblToolTip, SLOT(hide()));

*return* *true*;

}

*return* QWidget::event(pe);

}

*public*:

MyWidget(QWidget\* pwgt = 0) : QWidget(pwgt)

{

m\_plblToolTip = *new* QLabel;

m\_plblToolTip->setWindowFlags(Qt::*ToolTip*);

}

};

int main(int argc, char\*\* argv)

{

QApplication app(*argc*, argv);

MyWidget mw;

mw.setFixedSize(70, 70);

mw.setToolTip("<H1>My Tool Tip</H1>");

mw.show();

*return* app.exec();

}

1. Система помощи – навигатор HelpBrowser в Qt .

#ifndef HELPBROWSER\_H*//makes* *a* *window* *with* *help* *in* *a* *.htm* *file.* *mainwindow.cpp* *doesn't*

#define HELPBROWSER\_H*//necessarily* *has* *to* *have* *any* *code* *in* *itself*

#include <QTWidgets>

*class* HelpBrowser : *public* QWidget{

Q\_OBJECT

*public*:

HelpBrowser(*const* QString& strPath,

*const* QString& strFileName,

QWidget\* pwgt = 0

): QWidget(pwgt)

{

QPushButton\* pcmdBack = *new* QPushButton("<<");*//created* *widgets* *of* *buttons*

QPushButton\* pcmdHome = *new* QPushButton("Home");*//and* *pointers* *pcmdBack* *etc*

QPushButton\* pcmdForward = *new* QPushButton(">>");

QTextBrowser\* ptxtBrowser = *new* QTextBrowser;

connect(pcmdBack, SIGNAL(clicked()),*//connection* *of* *signals* *and* *slots*

ptxtBrowser, SLOT(home())*//apparently* *these* *signals* *and* *slots*

);*//are* *inherited*

connect(pcmdHome, SIGNAL(clicked()),*//configuring* *logistics* *inside*

ptxtBrowser, SLOT(home())

);

connect(pcmdForward, SIGNAL(clicked()),

ptxtBrowser, SLOT(forward())

);

connect(ptxtBrowser, SIGNAL(backwardAvailable(bool)),

pcmdBack, SLOT(setEnabled(bool))

);

connect(ptxtBrowser, SIGNAL(forwardAvailable(bool)),

pcmdForward, SLOT(setEnabled(bool))

);

ptxtBrowser->setSearchPaths(QStringList()<<strPath);

ptxtBrowser->*setSource*(QString(strFileName));

*//Layout* *setup*

QVBoxLayout\* pvbxLayout = *new* QVBoxLayout;

QHBoxLayout\* phbxLayout = *new* QHBoxLayout;

phbxLayout->addWidget(pcmdBack);

phbxLayout->addWidget(pcmdHome);

phbxLayout->addWidget(pcmdForward);

pvbxLayout->addLayout(phbxLayout);*//-book's* *example* *doesn't* *work-*

pvbxLayout->addWidget(ptxtBrowser);*//no,* *it* *works,* *I* *just*

setLayout(pvbxLayout);*//mixed* *up* *words* *as* *usual(addWidget* *instead* *of* *addLayout)*

}

};

#endif *//* *HELPBROWSER\_H*

#include "helpbrowser.h"

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

HelpBrowser helpBrowser(":/", "index.htm");*//helping* *window* *activation*

helpBrowser.resize(450, 350);

helpBrowser.show();

}

1. Простой список – класс QListWidget.

#include <QtWidgets>

int main(int argc, char\*\* argv)

{

QApplication app(*argc*, argv);

QStringList lst;

QListWidget lwg;

QListWidgetItem\* pitem = 0;

lwg.setIconSize(QSize(48, 48));

lst << "Linux" << "Windows" << "MacOSX" << "Android";

foreach(QString str, lst) {

pitem = *new* QListWidgetItem(str, &lwg);

pitem->setIcon(QPixmap(":/" + str + ".jpg"));

}

lwg.resize(125, 175);

lwg.show();

*return* app.exec();

}

1. Выбор элементов пользователем в списке QListWidget.

QListWidget::currentItem()//возвращает указатель на Item

adItem() – только для текста

InsertItem() – для всего остального

1. Режим пиктограмм, иерархические списки в Qt.

QStringList lst; *//-pictograms* *regime...* *doesn't* *work-*

QListWidget lwg;

QListWidgetItem\* pitem = *nullptr*;

lwg.setIconSize(QSize(48, 48));

lwg.setSelectionMode(QAbstractItemView::*MultiSelection*);

lwg.setViewMode(QListView::*IconMode*);

lst<<"Linux"<<"Windows"<<"MacOSX"<<"Android";

foreach(QString str, lst){

pitem = *new* QListWidgetItem(str, &lwg);

pitem->setIcon(QPixmap(":/" + str + ".jpg"));*//concatenates* *the* *thingies* *to* *get* *the*

pitem->setFlags(Qt::*ItemIsEnabled* |Qt::*ItemIsSelectable* |*//pics* *from* *the* *"/"* *prefix*

Qt::*ItemIsEditable* | Qt::*ItemIsDragEnabled*);

}

lwg.resize(150, 150);

lwg.show();

1. Таблицы, выпадающий список в Qt.

#include <QtWidgets>//chapter 11 TableWidget

int main(int argc, char\*\* argv)

{

*const* int n = 3;

QApplication app(*argc*, argv);

QTableWidget tbl(n, n);

QTableWidgetItem\* ptwi = 0;

QStringList lst;

lst << "First" << "Second" << "Third";

tbl.setHorizontalHeaderLabels(lst);

tbl.setVerticalHeaderLabels(lst);

*for* (int i = 0; i < n; ++i) {

*for* (int j = 0; j < n; ++j) {

ptwi = *new* QTableWidgetItem(QString("%1,%2").arg(i).arg(j));

tbl.setItem(i, j, ptwi);

}

}

tbl.resize(370, 135);

tbl.show();

*return* app.exec();

}

#include <QtWidgets>//chapter 11 ComboBox

*//* *----------------------------------------------------------------------*

int main(int argc, char\*\* argv)

{

QApplication app(argc, argv);

QComboBox cbo;

QStringList lst;

lst << "John" << "Paul" << "George" << "Ringo";

cbo.addItems(lst);

cbo.setEditable(*true*);

cbo.show();

*return* app.exec();

}

1. Операторы SQL: создание таблицы, добавление данных.

CREATE ТАВLЕ addressbook (

number INTEGER PRIМARYkeУ NOT NULL,

narne VARCНAR ( 15) ,

phone VARCНAR ( 12) ,

email VARCНAR(15)

);

INSERT INTO addressbook (numЬer, narne, phone, email)

VALUES(l, 'Piggy', '+49 631322187', 'piggy@mega.de');

INSERT INTO addressbook (numЬer, narne, phone, email)

VALUES(2, 'Kermit', '+49 631322181', 'kermit@mega.de');

1. Операторы SQL: выборка, изменение данных, удаление данных.

SELECT email

FROM addressbook

WHERE name = 'Piggy';

UPDATE addressbook

SET email = 'piggy@supennega.de'

WHERE name = 'Piggy';

DELETE FROМ addressbook//удаление строк

WHERE name = 'Piggy';

1. Три уровня классов модуля QtSql.

1. Уровень драйверов.

2. Программный уровень.

3. Уровень пользовательского интерфейса.

К первому уровню относятся классы для получения данных на физическом уровне. Это

такие классы, как QSqlDriver, QSqlDriverCreator<T\*>, QSqlDriverCreatorBase,

QSqlDriverPlugin И QSqlResult.

*602* Часть *VI.* Особые возможности *Qt*

Классы второго уровня предоставляют программный интерфейс для обращения к базе данных.

К классам этого уровня относятся следующие: QSqlDatabase, QSqlQuery, QSqlError,

QSqlField, QSqlindex и QSqlRecord.

Третий уровень предоставляет модели для отображения результатов запросов в представлениях

интервью. К этимклассам относятся: QSqlQueryModel, QSqlTaЫeModel и

QSqlRelationalTaЬleModel.

1. Алгоритм соединения с базой данный в Qt, чтение данных из базы.

Для соединения с базой данных прежде всего нужно активизировать драйвер. Для этого

вызывается статический метод QSqlDatabase: : addDatabase () (листинг 41.1 ). В него нужно

передать строку, обозначающую идентификатор драйвера СУБД

*static* bool createConnection()

{

QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName("addressbook");

db.setUserName("elton");

db.setHostName("epica");

db.setPassword("password");

*if* (!db.open()) {

qDebug() << "Cannot open database:" << db.lastError();

*return* *false*;

}

*return* *true*;

}

int main(int argc, char\*\* argv)//глава 41 SQLQuery Model типа для обычной БД

{

QApplication app(*argc*, argv);

*if* (!createConnection()) {

*return* -1;

}

QTableView view;

QSqlQueryModel model;

model.setQuery("SELECT phone, email "

"FROM addressbook "

"WHERE name = 'Piggy';"

);

*if* (model.lastError().isValid()) {

qDebug() << model.lastError();

}

view.*setModel*(&model);

view.show();

*return* app.exec();

}

int main(int argc, char\*\* argv)//а это уже для реляционной БД(с множественными связями)

{

QApplication app(*argc*, argv);

*if* (!createConnection()) {

*return* -1;

}

QTableView view;

QSqlRelationalTableModel model;

model.*setTable*("addressbook");

model.*setRelation*(0, QSqlRelation("status", "number", "married"));

model.*select*();

view.*setModel*(&model);

view.resize(450, 100);

view.show();

*return* app.exec();

}

int main(int argc, char\*\* argv)//SQLTableModel…

{

QApplication app(*argc*, argv);

*if* (!createConnection()) {

*return* -1;

}

QTableView view;

QSqlTableModel model;

model.*setTable*("addressbook");

model.*select*();

model.*setEditStrategy*(QSqlTableModel::*OnFieldChange*);

view.*setModel*(&model);

view.resize(450, 100);

view.show();

*return* app.exec();

}

int main(int argc, char\*\* argv){// Лмстмнr **41.2.** Испоnненме команд **SQL -** функция **llain ()** (файn **maln.cpp)**

QCoreApplication app(argc, argv);

*//* Соединяемся с менеджером баз данных

if ( ! createConnection () )

return -1;

*//* Создаем базу

QSqlQuery query;

QString str = "CREATE ТАВLЕ addressbook ( "

"numЬer INTEGER PRIМARY КЕУ NOT NULL, "

"name VARCНAR(l5), "

"phone VARCНAR ( 12) , "

"email VARCНAR(l5) "

") ;";

if (!query.exec(str)) {

qDebug() << "Unable to create а tаblе";

*//* Добавляем данные в базу

QString strF =

"INSERT INTO addressbook (numЬer, name, phone, email) "

"VALUES(%1, '%2', '%3', '%4');";

str strF.arg("l")

. arg ( "Piggy")

.arg("+49 631322187")

.arg("piggy@mega.de");

if (!query.exec(str)) {

qDebug() << "Unable to make insert operation";

str strF.arg("2")

.arg("Kermit")

.arg("+49 631322181")

.arg ( "kermit@mega.de");

if (!query.exec(str)) {

qDebug () << "Unable to make insert operation";

if ( ! query. ехес ( “ SELECT \* FROM addressbook; “) )

qDebug() << “ Unable to execute query – exiting”;

return 1;

*//* Считываем данные из базы

QSqlRecord rec = query.record();

Int nNumber = 0;

QString strName;

QString strPhone;

QString strEmail;

while (query.next())

nNumber query.value(rec.indexOf( “ number “ )) .toInt();

strName query.value(rec.indexOf( “name “ )) .toString();

strPhone query.value(rec.indexOf( “phone” )) .toString();

strEmail query.value(rec.indexOf( “email “ )) .toString();

qDebug() << nNumber << " 11 << strName << “ ;\t"

<< strPhone << “;\t” << strEmail;

return 0;

Листинг 41.3. Использование модели опроса. Файл main.cpp. Функция main() int

main(int argc, char\*\* argv) {

QApplication app(argc, argv);

if (!createConnection()) { return -1; }

QTableView view;

QSqlQueryModel model;

model.setQuery("SELECT phone, email " "FROM addressbook " "WHERE name = 'Piggy';" );

if (model.lastError().isValid()) { qDebug() << model.lastError(); } view.setModel(&model); view.show(); return app.exec();

}

1. Класс QSqlQuery: запросы, навигация.
2. Способы построения запроса INSERT.

Вставка новых записей Для того чтобы вставить в таблицу новые записи, надо вызвать метод insertRow(), в первом параметре этого метода нужно указать индекс строки, во втором — количество новых строк, а затем вызовами методов setData() в новые строки внести необходимые данные. Запомните, что при использовании стратегий обновления OnFieldChange и OnRowChange за один раз можно вставить только одну строку. Итак, в нашу адресную книгу внесем еще одного героя: model.insertRows(0, 1);

model.setData(model.index(0, 0), 4);

model.setData(model.index(0, 1), "Sam");

model.setData(model.index(0, 2), "+49 63145476576");

model.setData(model.index(0, 3), "sam@mega.de"); if (!model.submitAll()) { qDebug() << "Insertion error!";}

Реляционная модель Реляционная модель представляет собой более высокий уровень реализации, чем тот, который предоставляет табличная модель. Она обладает механизмами связывания таблиц с помощью первичных (primary) и/или вторичных ключей (foreign keys). Это позволяет модели искать информацию сразу в нескольких таблицах и отображать их в одной. Класс для реляционной модели в Qt это QSqlRelationalTableModel. Он унаследован от только что рассмотренного класса QSqlTableModel. Этот класс к унаследованным методам добавляет только три новых метода, которые предназначены лишь для работы со связями таблиц: relationModel(), relation() и setRelation(). Для демонстрации дополним нашу базу данных "addressbook" еще одной таблицей "status", которая будет содержать информацию о том, женат/замужем контакт или нет.

CREATE TABLE status ( number INTEGER PRIMARY KEY NOT NULL, married VARCHAR(5) " );

Внесем туда данные: INSERT INTO status (number, married) VALUES(1, 'YES'); INSERT INTO status (number, married) VALUES(2, 'NO');

Теперь выполним отображение данных из обеих таблиц, как это показано на рис. 4



1. Концепция интервью в Qt: показ базы данных, выборка.
2. Характеристики и методы класса QPainter.

Класс QPainter, определенный в заголовочном файле QPainter, является исполнителем команд

рисования. Он содержит массу методов для отображения линий, прямоугольников,

окружностей и др. Рисование осуществляется на всех объектах классов, унаследованных от

класса QPaintDevice (рис. 18.3). Это означает, что то, что отображается контекстом рисования

одного объекта, может быть точно так же отображено контекстом и другого объекта.

Чтобы использовать объект QPainter, необходимо передать ему адрес объекта контекста, на

котором должно осуществляться рисование. Этот адрес можно передать как в конструкторе,

так и с помощью метода QPainter: : begin (). Смысл метода begin () состоит в том, что он

позволяет рисовать на одном контексте несколькими объектами класса QPainter. При использовании

метода begin () нужно по окончании работы с контекстом вызвать метод

QPainter: : end (), чтобы отсоединить установленную этим методом связь с контекстом рисования,

давая возможность для рисования другому объекту



#include <QtWidgets>

*//* *----------------------------------------------------------------------*

QLabel\* **lbl**(*const* QPainter::CompositionMode& mode)

{

QLabel\* plbl = *new* QLabel;

plbl->setFixedSize(100, 100);

QRect rect(plbl->contentsRect());

QPainter painter;

QImage sourceImage(rect.size(), QImage::*Format\_ARGB32\_Premultiplied*);

sourceImage.fill(QColor(0, 0, 0, 0));

painter.begin(&sourceImage);

painter.setRenderHint(QPainter::*Antialiasing*, *true*);

painter.setBrush(QBrush(QColor(0, 255, 0)));

painter.drawPolygon(QPolygon() << rect.bottomLeft()

<< QPoint(rect.center().x(), 0)

<< rect.bottomRight()

);

painter.end();

QImage resultImage(rect.size(), QImage::*Format\_ARGB32\_Premultiplied*);

painter.begin(&resultImage);

painter.setRenderHint(QPainter::*Antialiasing*, *true*);

painter.setCompositionMode(QPainter::*CompositionMode\_SourceOver*);

painter.setPen(QPen(QColor(0, 255, 0), 4));

painter.setBrush(QBrush(QColor(255, 0, 0)));

painter.drawEllipse(rect);

painter.setCompositionMode(mode);

painter.drawImage(rect, sourceImage);

painter.end();

plbl->setPixmap(QPixmap::fromImage(resultImage));

*return* plbl;

}

*//* *----------------------------------------------------------------------*

int main(int argc, char\*\* argv)

{

QApplication app(*argc*, argv);

QWidget wgt;

*//Layout* *Setup*

QGridLayout\* pgrd = *new* QGridLayout;

pgrd->addWidget(lbl(QPainter::*CompositionMode\_Source*), 0, 0);

pgrd->addWidget(*new* QLabel("<CENTER>Source</CENTER>"), 1, 0);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_SourceOver*), 0, 1);

pgrd->addWidget(*new* QLabel("<CENTER>SourceOver</CENTER>"), 1, 1);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_SourceIn*), 0, 2);

pgrd->addWidget(*new* QLabel("<CENTER>SourceIn</CENTER>"), 1, 2);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_SourceOut*), 0, 3);

pgrd->addWidget(*new* QLabel("<CENTER>SourceOut</CENTER>"), 1, 3);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_SourceAtop*), 0, 4);

pgrd->addWidget(*new* QLabel("<CENTER>SourceAtop</CENTER>"), 1, 4);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_Clear*), 0, 5);

pgrd->addWidget(*new* QLabel("<CENTER>Clear</CENTER>"), 1, 5);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_Destination*), 2, 0);

pgrd->addWidget(*new* QLabel("<CENTER>Destination</CENTER>"), 3, 0);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_DestinationOver*), 2, 1);

pgrd->addWidget(*new* QLabel("<CENTER>DestinationOver</CENTER>"), 3, 1);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_DestinationIn*), 2, 2);

pgrd->addWidget(*new* QLabel("<CENTER>DestinationIn</CENTER>"), 3, 2);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_DestinationOut*), 2, 3);

pgrd->addWidget(*new* QLabel("<CENTER>DestinationOut</CENTER>"), 3, 3);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_DestinationAtop*), 2, 4);

pgrd->addWidget(*new* QLabel("<CENTER>DestinationAtop</CENTER>"), 3, 4);

pgrd->addWidget(lbl(QPainter::*CompositionMode\_Xor*), 2, 5);

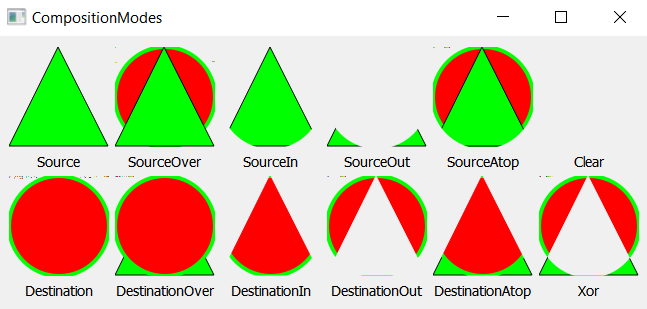
pgrd->addWidget(*new* QLabel("<CENTER>Xor</CENTER>"), 3, 5);

wgt.setLayout(pgrd);

wgt.show();

*return* app.exec();

}



1. Класс QCanvas.
2. OpenGL в Qt.
3. Класс QDate : методы.
4. Класс QDate : создание своего формата времени, изменение времени.
5. Класс QTime: методы.
6. Класс QTime: вычисление времени работы функции.
7. Класс: QDateTime.
8. Класс QTime: подобие таймера.
9. Таймер, события таймера.
10. Применение таймера.
11. Одноразовый таймер, таймер с функцией connect().
12. Класс QBasicTimer.
13. Формат XML, его применение.
14. Структура XML-файла, теги и их особенность.
15. Модель DOM: иерархия документа.
16. Модель DOM: чтение XML-документа.
17. Модель DOM: создание и запись XML-документа.
18. Модель SAX: иерархия документа.
19. Модель SAX: чтение XML-документа.
20. Модель SAX: создание и запись XML-документа.
21. Основные классы модуля QtWebKit, свойства класса QWebView.
22. Методы класса QWebView, изменение настроек виджета WebView с помощью класса QWebSettings.
23. Класс QWebHistory, отображение истории работы с браузером.
24. Возможности класса QNetworkAccessManager, взаимодействие между QtNetWorkAccessManager и вэб-ресурсом.
25. Сигнал finished([QNetworkReply\*](http://doc.crossplatform.ru/qt/4.7.x/qnetworkreply.html) reply): вызов, параметры, получение ответа.
26. QtNetWorkAccessManager: Формирование POST и GET запросов.
27. Класс QCache, механизм кэширования.
28. Контекстно-независимое представление растровых изображений в Qt.
29. Контекстно-зависимое представление растровых изображений в Qt.
30. Использование каскадных стилей документа в Qt Creator.
31. Сохранение настроек приложения.
32. Область оповещений на рабочем столе.