دوره آموزشی برنامه نویسی C++ در لینوکس

برنامه‌نویسی C++ در کیوت Qt

بخش ۱ : معرفی

فهرست

مقدمه

- چهارچوب توسعه‌ی نرم‌افزار کیوت (Qt Application Development Platform) یک ابزار قدرتمند برای برنامه‌نویسی C++ است.
  - دارای قابلیت چندسکویی (Platform Independent)
    - Write Once, Compile Everywhere!
  - رایگان و متن باز!
    - تحت مجوز GPL
  - یادگیری آن آسان و دارای مستندات قدرتمند است.
  - در سیستم عامل‌های غیر از ویندوز، کیوت جامع‌ترین ابزار برنامه‌نویسی کاربردی است.
  - تاکنون برنامه‌های تجاری و رایگان بسیاری با کیوت نوشته شده است:
    - KDE، Opera، Google Earth، Skype، VLC، Maya و Mathematica

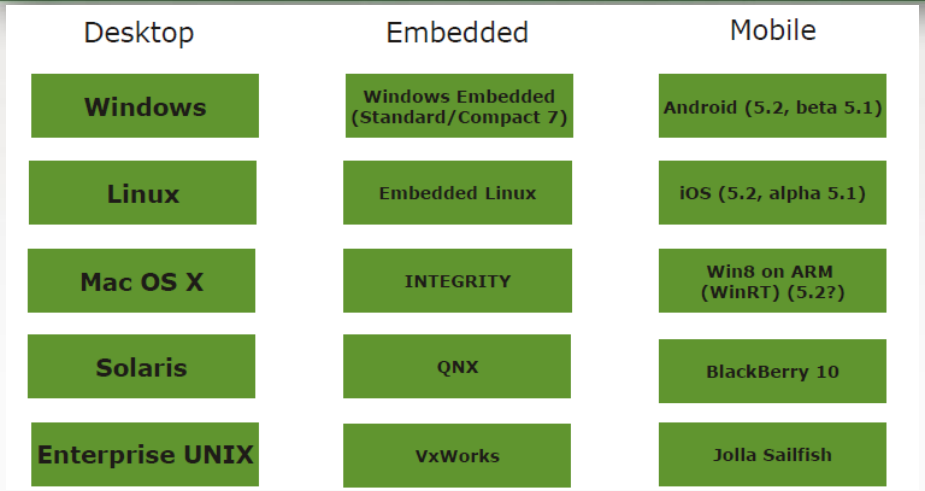
- در این اسلاید آموزشی قصد داریم، تا با این ابزار قدرتمند آشنا شده و از آن برای توسعه‌ی نرم‌افزار استفاده کنیم.

تاریخچه

- کیوت برای اولین بار در ماه می ۱۹۹۵ توسط شرکت Trolltech (نروژ) منتشر شد.
  - با دو لیسانس تجاری و متن باز
    - به این معنا که هم برای کاربردهای متن باز و هم برای کاربردهای تجاری می‌تواند مورد استفاده قرار بگیرد.
- در سال ۲۰۰۸ این شرکت توسط نوکیا خریداری شد.
  - تغییر نام محصول به Qt Development Frameworks
- در سال ۲۰۱۲ یک شرکت فنلاندی (Digia) کیوت را از نوکیا خریداری نمود.
- هم‌زمان توسعه‌ی نرم‌افزاری متن باز کیوت نیز ادامه دارد.
- جدیدترین نسخه‌ی کیوت در حال حاضر نسخه‌ی ۵.۱ است.
  - مورد مطالعه در این اسلاید.
- برای مشاهده‌ی تغییرات به وجود آمده طی زمان در نسخه‌های کیوت، به فولدر زیر مراجعه نمایید.

Qt\Tutorials\Qt5vsOlderVersions

ویژگی‌های کیوت



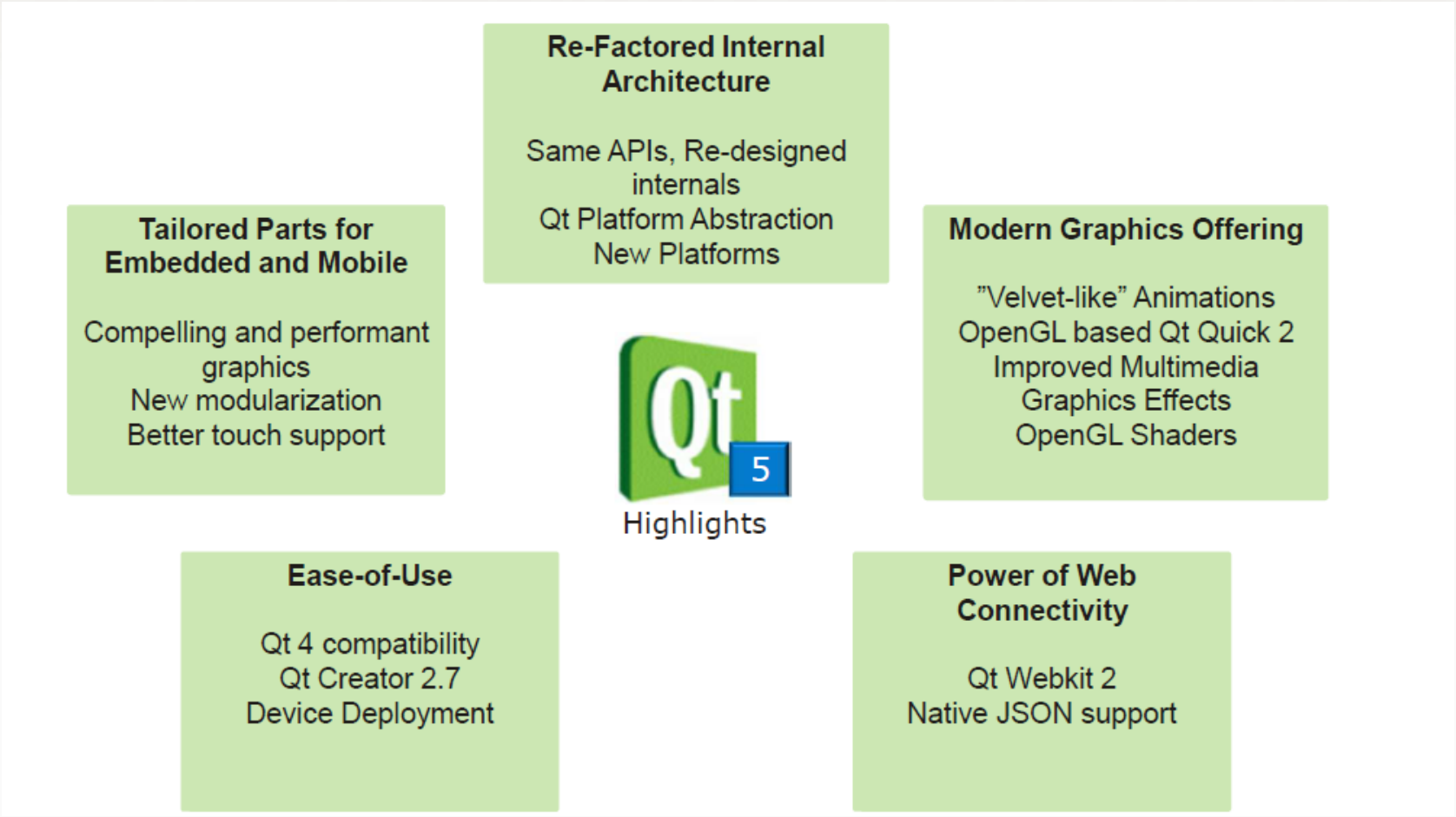
- چندسکویی
  - تاکنون، برنامه‌های نوشته شده در کیوت قابلیت کامپایل بر روی هر یک ازسیستم عامل‌های مقابل را دارند.
- حمایت از بسیاری از کتابخانه‌های مهم برای برنامه‌نویسی C++
  - QtCore: Threads, Processes, Data types, Containers, File I/O
  - QtNetwork: TCP/UDP, HTTP, FTP, SSL
  - QtWebkit ،QtSql
  - Qt Serial Port (new in 5.1)
  - ...
- رایگان، متن باز، قدرتمند و با مستندات بسیار خوب.

ویژگی‌های کیوت

- چندسکویی

ویژگی‌های کیوت

- ویژگی‌های کیوت نسخه‌ی ۵



ویژگی‌های کیوت

- ویژگی‌های Qt 5.1
  - بیش از ۳۰۰۰ خطای موجود در نسخه 5.0 برطرف شده است.
  - ابزار Qt Quick برای کامپیوترهای دسکتاپ هم قدرت بالاتری دارد (Qt Quick 2)
  - سیستم عامل‌های Android و IOS هم تحت پوشش قرار گرفته‌اند.
  - اضافه شدن ماژول‌های جدید به کیوت:
    - Qt Sensors
    - Qt Serial Port
    - X11 Extras
    - Wayland
  - سازگاری با Qt 4.x (با تعدادی محدودی استثنا)
  - حمایت از C++11
  - واسط کاربری به‌روز و جدید برای برنامه‌ها
- برای مطالعه‌ی بیشتر بر روی ویژگی‌های کیوت:

Qt\Tutorials\Qt5vsOlderVersions

دانلود و نصب کیوت

- در اینجا دانلود و نصب کیوت در محیط لینوکس را برای دو نسخه توضیح خواهیم داد:
  - Qt 4 SDK :
    - ❖ نصب کیوت برای این نسخه بسیار ساده است و تنها به اجرای دستور زیر نیازمند می‌باشد.
      - دستور را در محیط ترمینال، در مکانی دلخواه، تایپ نموده و اجرا نمایید.

```
sudo apt-get install qt-sdk
```

  - Qt 5.1.0 : آخرین نسخه تاکنون
    - ❖ دانلود نسخه مناسب از http://qt-project.org/downloads **و یا** استفاده از مسیر زیر:

```
Qt\Setup\qt-linux-opensource-5.1.0-x86_64-offline.run
```

❖ اجرای دستور زیر در محیط ترمینال برای ایجاد اجازه‌ی اجرا برای فایل

```
chmod 777 Qt\Setup\qt-linux-opensource-5.1.0-x86_64-offline.run
```

❖ اجرای دستور زیر در محیط ترمینال برای اجرای فایل دانلود شده و ورود به محیط نصب

```
./qt-linux-opensource-5.1.0-x86_64-offline.run
```

❖ نکته: برای راحتی کار با کیوت، نرم‌افزار را بدون دستور sudo و در فولدر home از سیستم خود نصب نمایید.

دانلود و نصب کیوت

- نصب کیوت – ادامه
  - Qt 5.1.0 – ادامه
    - ❖ نصب کامپایلرهای مورد نیاز با اجرای دستوری شبیه به زیر (تنها بخش قرمز ضروری است).

```
sudo apt-get install build-essential perl python git
```

    - ❖ اجرای دستور زیر برای نصب OpenGL مخصوص لینوکس (mesa).

```
sudo apt-get install mesa-common-dev
```

    - ❖ اجرای دستور زیر برای رفع مشکلات مرتبط با نشناختن OpenGL توسط کامپایلر.

```
sudo apt-get install libglu1-mesa-dev
```

    - ❖ اجرای دستور زیر تا پس از آن، Qt5 در محیط لینوکس به صورت مستقل شناخته شود.

```
sudo apt-get install qt5-default
```

  - اکنون بایستی کیوت به‌درستی نصب شده باشد.
    - ❖ برای تست، می‌توانید نمونه پروژه‌های موجود در کیوت را اجرا نمایید.

ابزارها

- مهم‌ترین ابزارهای توسعه‌ی نرم‌افزار کیوت موارد زیر می‌باشند:
  - Qt Creator
  - Qt Designer
  - qmake
  - moc
  - uic
  - سایر ابزارها
    - ❖ Qt Assistant
    - ❖ Qt Linguist

ابزارها

- Qt Creator
  - محیط اصلی چهارچوب توسعه‌ی نرم‌افزاری کیوت را تشکیل می‌دهد.
  - جهت مدیریت، اشکال‌زدایی، کامپایل، پیوند، ساخت و اجرای کلیه‌ی پروژه‌های کیوت قابل استفاده است.
    - ❖ سایر ابزارهای کیوت را می‌توان در محیط Qt Creator مشاهده نموده یا از آن‌ها استفاده کرد.
- Qt Designer
  - یک ابزار جهت طراحی و ساخت واسط‌های گرافیکی کاربر (GUIs) توسط ویجت‌های کیوت است.
    - ❖ ویجت‌ها (Window Gadgets) کلاس‌هایی از کیوت هستند که از QWidget مشتق شده‌اند.
  - می‌توان پنجره‌ها و دیالوگ‌ها را به صورتی قابل مشاهده طراحی، تنظیم و با سبک‌ها و رزولوشن‌های مختلف آزمایش نمود.
  - ویجت‌ها و فرم‌های ساخته شده در Designer به راحتی قابل ارتباط و استفاده در کد برنامه هستند.
    - ❖ تمامی ویژگی‌های طراحی شده در Designer را می‌توان به صورت پویا در کد برنامه تغییر داد.
  - ویژگی‌هایی نظیر widget promotion و custom plugins اجازه می‌دهند تا کلاس‌هایی جدید را به Designer اضافه کنیم.

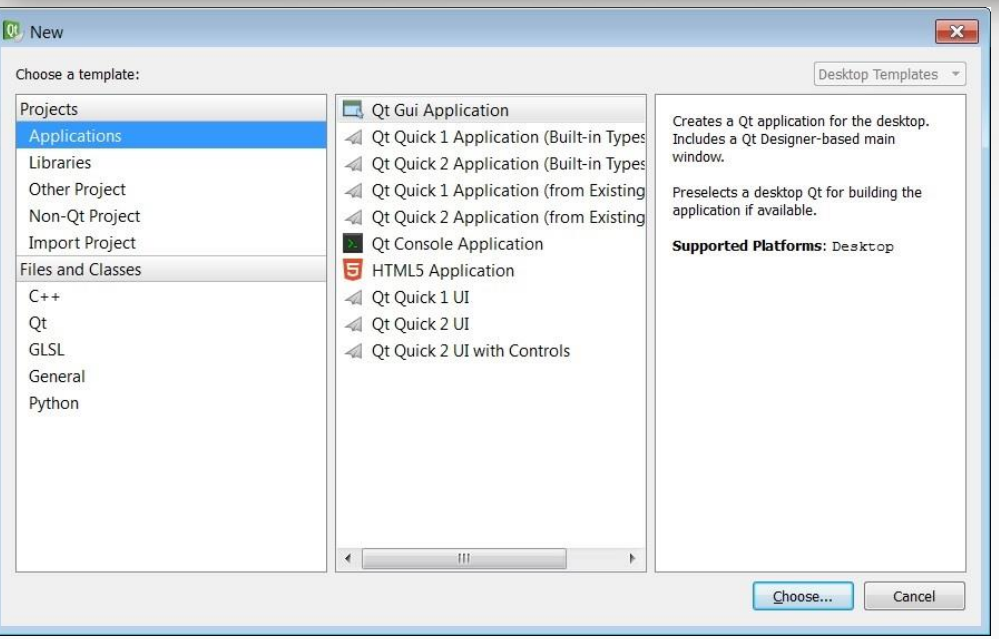
ابزارها

- qmake
  - به فرآیند ساخت (build) در پروژه‌های تولیدی، بین سکوهای متفاوت کمک می‌کند.
  - به صورت خودکار اقدام به تولید makefile برای برنامه می‌کند.
  - از qmake می‌توان برای هر پروژه‌ی نرم‌افزاری (چه کیوت و چه دیگر پروژه‌ها) استفاده نمود.
  - qmake برای تولید makefile از اطلاعات درون فایل پروژه (pro. در پروژه‌های کیوت) استفاده می‌کند.
  - در پروژه‌های کیوت، قوانین ساخت برای moc و uic را به صورت خودکار به کار می‌گیرد.
- moc (Meta-Object Compiler)
  - برای رسیدگی به افزوده‌های کیوت بر C++ (مانند سیگنال‌ها و اسلات‌ها) کاربرد دارد.
    - ❖ کد C++ تولید شده توسط moc بایستی همراه با پیاده‌سازی کلاس، کامپایل و build شود.
- uic (User Interface Compiler)
  - یک فایل واسط کاربری را خوانده و سرآیند C++ منطبق با آن را تولید می‌نماید.
    - ❖ فایل واسط کاربری نتیجه‌ی طراحی کاربر در محیط Designer است (یک فایل XML با پسوند ui.).

ابزارها

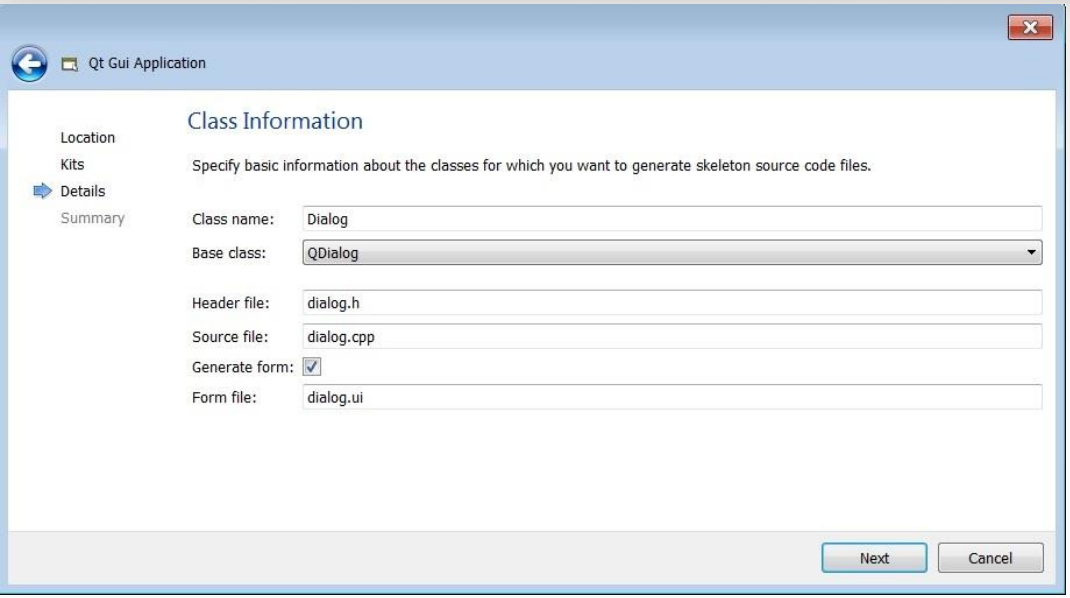
- سایر ابزارها
  - Qt Assistant
    - ابزاری برای مشاهده‌ی مستندات آنلاین کیوت در قالب فایل است.
    - می‌توان از Assistant به عنوان راهنمای کاربر در برنامه‌های خود استفاده نمود.
      - مستندات شخصی و Assistant دستکاری شده را می‌توان به عنوان بخشی از برنامه‌ی خود به نمایش گذاشت.
  - Qt Linguist
    - کیوت حمایت قدرتمندی برای ترجمه‌ی برنامه‌های کاربردی Qt C++ و Qt Quick به زبان‌های غیر انگلیسی فراهم کرده است.
    - مدیران انتشار (Release Managers)، مترجمان (Translators) و برنامه‌نویسان (Developers) می‌توانند از ابزارهای کیوت برای وظایف خود بهره ببرند.
      - مدیران انتشار می‌توانند از ابزار lupdate برای «هم‌زمان‌سازی کد برنامه و ترجمه‌ها» و از ابزار lrelease برای ساختن «مترجم‌های حین اجرا» استفاده نمایند.
      - مترجمان می‌توانند از ابزار Qt Linguist برای ترجمه‌ی متن‌ها استفاده نمایند.
      - برنامه‌نویسان بایستی برنامه‌هایی بنویسند که قادر به استفاده از متن‌های ترجمه شده است.

ساخت پروژه در محیط Creator



- مرحله‌ی اول: ساخت پروژه‌ی جدید
  - وارد محیط creator شوید.
  - File->New File or Project
  - منوی باز شده که در شکل مشخص است، انواعی از پروژه‌های ممکن برای ساخت را دارد.
  - مرسوم‌ترین حالت:
    - Qt Gui Application را انتخاب کنید.
  - نام و محل ذخیره‌سازی پروژه‌ی خود را تعیین نمایید.
    - برای مثال نام پروژه helloworld و محل ذخیره‌سازی فولدر /Qt/Codes/.../ است.
  - سپس گزینه‌ی next را بزنید.

ساخت پروژه در محیط Creator



- مرحله‌ی اول – ادامه
  - در صفحه‌ی جاری بایستی برای کلاس اصلی برنامه، نام و نوع انتخاب نمایید.
  - در صورتی که generate form را انتخاب نمایید، برای کلاس اصلی یک فرم XML (قابل دستکاری از طریق Designer) فراهم می‌نماید.
  - گزینه‌ی next را بزنید.
  - صفحه‌ی بعد گزارشی از ایجاد پروژه به همراه فایل‌های افزوده به پروژه را نمایش می‌دهد.
  - بر روی Finish کلیک نمایید.
- اکنون ساخت پروژه‌ی جدید به پایان رسیده است. در صورت اجرا دیالوگ اصلی برنامه (یک دیالوگ خالی) به نمایش درمی‌آید.

ساخت پروژه در محیط Creator

```
QT += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT +=
widgets
TARGET = helloworld TEMPLATE = app
SOURCES += main.cpp\ dialog.cpp
HEADERS += dialog.h
FORMS += dialog.ui
```

- مرحله‌ی دوم – اصلاح فایل helloworld.pro
  - محتویات فایل به صورت مقابل است.
  - core و gui به ترتیب نشان‌گر این هستند که پروژه قصد دارد از توابع هسته و توابع واسط گرافیکی استفاده نماید.
    - در صورتی که قرار است از کلاس‌های OpenGL کیوت نیز استفاده نمایید:

```
QT += core gui opengl
```

    - در صورتی که قصد دارید از کلاس‌های شبکه‌ی کیوت نیز استفاده نمایید:

```
QT += core gui network
```

    - در صورتی که قصد استفاده از کلاس‌های ویجت در برنامه خود دارید:

```
QT += core gui widgets
```

  - برای اینکه از بتوانید از کتابخانه‌ای غیر از کتابخانه‌ی جاری کیوت در برنامه‌ی خود استفاده نمایید:
    - بر روی ریشه‌ی پروژه‌ی خود (helloworld) راست کلیک کرده و گزینه‌ی Add Library را انتخاب نمایید.
    - در منوی جاری گزینه‌ی External Library را انتخاب نموده و next بزنید.
    - در خط اول مسیر فایل کتابخانه (در لینوکس فایل so.) و در خط دوم آدرس سرآیندهای مربوط به کتابخانه را وارد نمایید.
    - گزینه‌های دیگر را متناسب با نیاز خود انتخاب کرده و سپس بر روی گزینه‌ی next کلیک نمایید.
    - اکنون کتابخانه به پروژه‌ی شما افزوده شده است. می‌توانید تغییرات ایجاد شده را در فایل pro. بررسی نمایید.

ساخت پروژه در محیط Creator

- مرحله‌ی سوم – افزودن کلاس‌های مورد نیاز
  - بر روی ریشه‌ی پروژه راست کلیک کرده و گزینه‌ی Add New یا Add Existing را انتخاب نمایید.
  - فایل یا کلاس مورد نیاز خود را انتخاب کنید.
- مرحله‌ی چهارم – طراحی gui در Designer
  - بر روی فایل ui. دوبارکلیک کرده تا وارد محیط Designer شوید.
  - تغییرات خود را اعمال نموده و ذخیره کنید.
- مرحله‌ی پنجم – استفاده از اشیای تعریف شده در gui در کد برنامه
  - اضافه کردن دستور مقابل در ابتدای فایل dialog.h

```
#include “ui_dialog.h”
```

  - تعریف کلاس Ui::Dialog به عنوان پایه‌ای برای کلاس Dialog

```
class Dialog : public QDialog, public Ui::Dialog {
        . . .
}
```

  - افزودن دستور روبه‌رو در ابتدای سازنده‌ی کلاس Dialog :

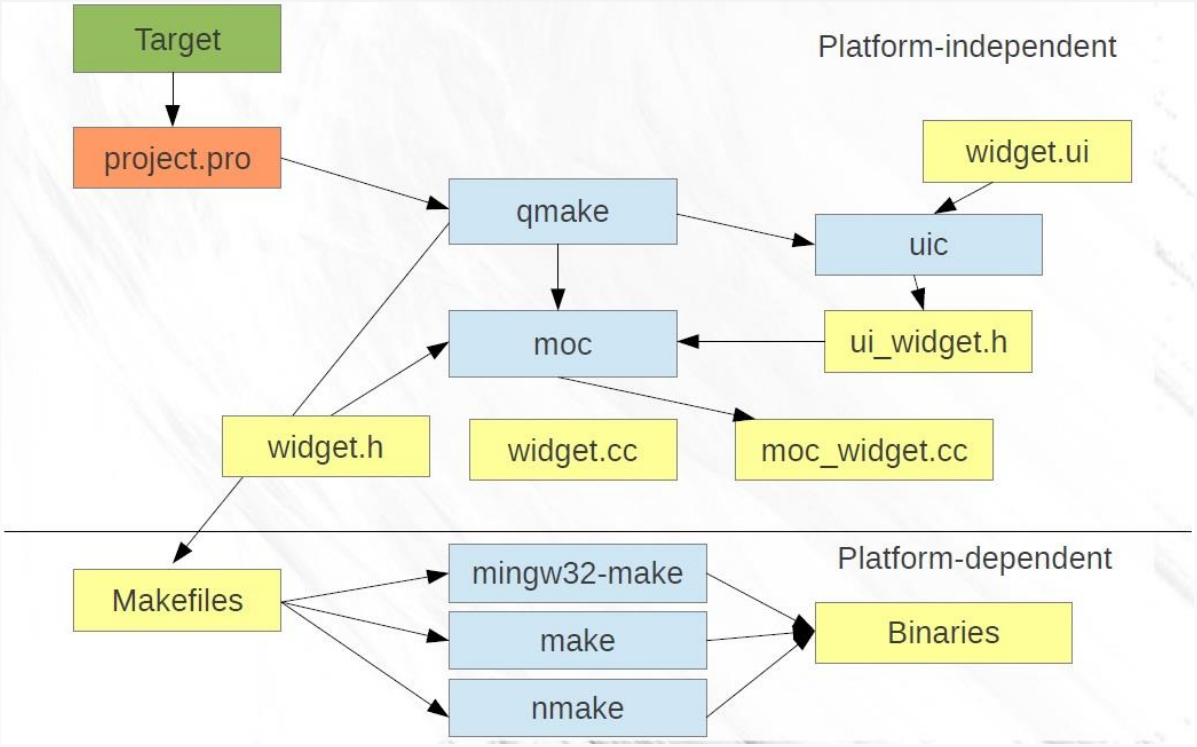
```
setupUi(this);
```

  - روش دوم: تعریف اشاره‌گری به شئ Ui::Dialog در کلاس Dialog و کار با آن مانند سایر اشاره‌گرها است.

```
Ui::Dialog* ui = new Ui::Dialog;
ui->setupUi(this);
```

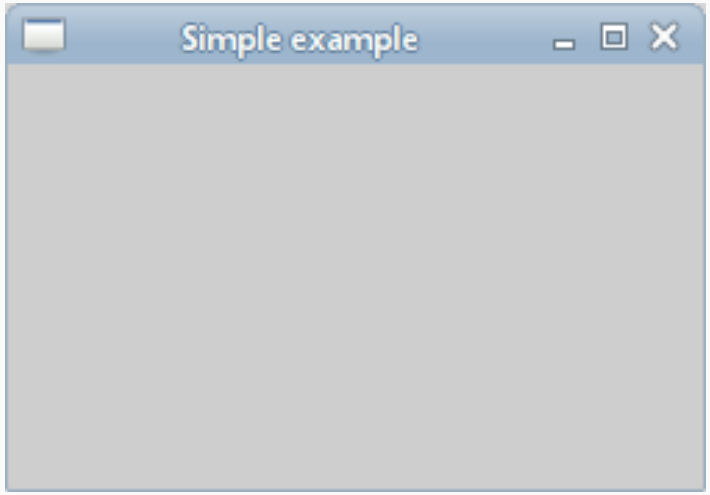
ساخت پروژه در محیط Creator

- فرآیند ساخت : به شکل زیر توجه کنید.
  - کاربر تنها حق نوشتن یا ویرایش فایل‌های زیر را دارد:
    - widget.h
    - widget.cc
    - widget.ui
    - project.pro
    - سایر فایل‌های شکل بهتر است از سوی کاربر دستکاری نشوند.



Hello Qt!

```
#include <QApplication>
#include <QWidget>
int main(int argc, char *argv[]) {
        QApplication app(argc, argv);
        QWidget window;
        window.resize(250, 150);
        window.setWindowTitle("Simple
example");
        window.show();
        return app.exec();
}
```



- روشی دیگر برای ساخت و اجرای برنامه‌ها بدون نیاز به Creator
  - برنامه‌ی روبه‌رو را در یک فایل بنویسید.
    - ❖ فایل hello.cpp در فولدر /hello/.../
    - ❖ اجرای برنامه، خروجی مشابه شکل زیر را تولید خواهد کرد.
- کامپایل و اجرای برنامه‌ی نوشته‌شده خارج از Creator
  - در محیط ترمینال به فولدر /hello/.../ بروید.
  - دستورات زیر را به ترتیب اجرا نمایید.
    - ❖ ساخت فایل pro. شامل تمامی فایل‌های درون پوشه

```
qmake –project
```

    - ❖ ساخت makefile

```
qmake hello.pro
```

    - ❖ ساخت (build)

```
make
```

    - ❖ اجرای برنامه

```
./hello
```

Hello Qt!

- توضیحی کوتاه بر خط‌های برنامه

```
#include <QApplication>
#include <QWidget>
```

- افزودن سرآیندهای مورد نیاز

```
QApplication app(argc, argv);
```

- هر برنامه‌ی کاربردی کیوت دارای شئ QApplication است
  - ❖ به جز برنامه‌های کنسول

```
QWidget window;
```

- ویجت اصلی برنامه

```
window.resize(250, 150);
window.setWindowTitle("Simple example");
window.show();
```

- تنظیم پنجره (ویجت) اصلی برنامه
  - ❖ تغییر سایز، تنظیم عنوان، نمایش

```
return app.exec();
```

- سپردن کنترل برنامه به کیوت

Hello Qt!

```
QDesktopWidget *desktop = QApplication::desktop();
int screenWidth = desktop->width();
int screenHeight = desktop->height();
x = (screenWidth - WIDTH)/2;
y = (screenHeight - HEIGHT)/2;
window.resize(WIDTH, HEIGHT);
window.move( x, y );
window.setToolTip("QWidget");
window.setWindowIcon(QIcon("web.png"));
QFrame *frame = new QFrame(this);
frame->setFrameStyle(QFrame::Box);
frame->setCursor(Qt::SizeAllCursor);
QGridLayout *grid = new QGridLayout(this);
grid->addWidget(frame, 0, 0);
setLayout(grid)
QPushButton *quit = new QPushButton("Quit", this);
quit->setGeometry(50, 40, 75, 30);
connect(quit, SIGNAL(clicked()), qApp, SLOT(quit()));
```

- برخی دستورات ساده‌ی قابل افزایش به برنامه
  - تنظیم ابعاد و مکان ویجت اصلی
    - محاسبه‌ی طول و عرض صفحه‌ی نمایش
    - محاسبه‌ی مکان قرارگیری صفحه‌ی ما
    - تغییر ابعاد و انتقال به مکان مورد نظر
  - افزودن یک راهنما (Hint) در مورد یک ابزار
  - افزودن آیکون برای برنامه‌ی نوشته شده
  - ساختن یک ویجت از نوع فریم
    - افزودن استایل و cursor برای آن
  - افزودن فریم مورد نظر به layout اصلی
  - افزودن یک دکمه برای خروج
  - متصل کردن دکمه به خروج
    - سیگنال کلیک دکمه و اسلات خروج از برنامه

مهمترین اشیاء و ویجت‌ها

کلاس‌های پایه

- QObject
- QApplication
- QWidget
- QDialog
- QMainWindow
  - QMenu
  - QAction
  - QToolBar
  - QMenuBar
  - …

برخی کلاس‌های کاربردی

- QTextStream
- QFile
- QList
- QDir
- QImage
- QTime
- QString
- QPainter

برخی از ویجت‌ها

- QLabel
- QSlider
- QSpinBox
- QLineEdit
- QStatusBar
- QCheckBox
- QListWidget
- QPixmap
- QSplitter
- QTableWidget

کلاس‌های پایه

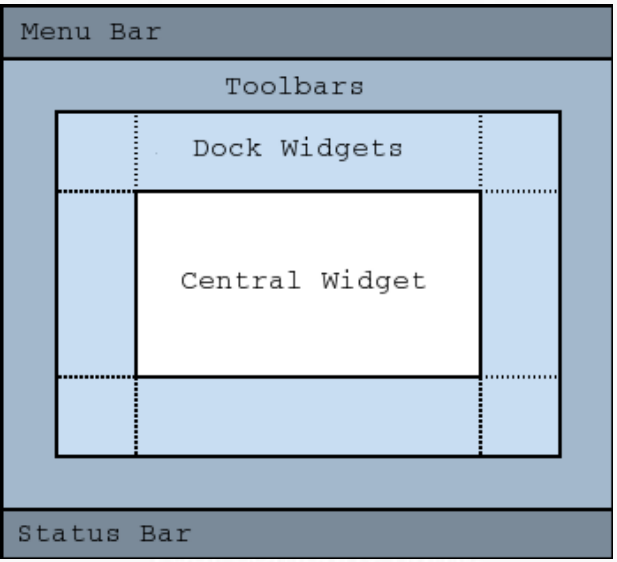
- QObject
  - کلاس پایه‌ی تمامی اشیای کیوت می‌باشد.
  - تنها برای QObjectها قابلیت استفاده از ارتباطات Signal-Slot وجود دارد.
  - هنگام ساختن یک QObject جدید:
    - همواره اجازه‌ی null (0) بودن parent را فراهم سازید (حالت پایه).
    - سعی کنید سازنده‌ای داشته باشید که تنها parent را به عنوان آرگومان قبول کند.
    - چندین سازنده داشته باشید تا مجبور نشوید هنگام فراخوانی، null (0) را به عنوان آرگومان پاس بدهید.
    - همواره parent را اولین آرگومانی از سازنده در نظر بگیرید که دارای مقدار پایه است.
- QApplication
  - جریان کنترلی و تنظیمات اصلی برنامه‌های کاربردی GUI را مدیریت می‌کند.
  - برای هر برنامه‌ی کاربردی GUI در کیوت، یک و فقط یک شئ QApplication وجود دارد.
    - مهم نیست که برنامه دارای صفر، یک، دو یا تعداد بیشتری پنجره باشد.
  - برای برنامه‌هایی از کیوت که بر پایه‌ی QWidget نیستند می‌توان از QGuiApplication بهره برد.

کلاس‌های پایه

- QWidget
  - از QObject مشتق شده است.
  - تمامی عناصر نمایشی بایستی QWidget را به عنوان کلاس پایه‌ی (Base Class) خود داشته باشند.
    - ❖ این اجزا می‌توانند به صورت تودرتو تعریف شوند (با مشخص کردن ساختار توسط آرگومان parent در سازنده‌ی QWidget).
    - ❖ هنگامی که یک parent برای یک QWidget در نظر گرفته می‌شود، پاک کردن حافظه‌ی گرفته شده توسط شی (پس از نابودی اش) خودبه‌خود توسط parent انجام خواهد شد و نیازی به دخالت مستقیم کاربر نیست.
    - ❖ اگر QWidget دارای parent نباشد (=0)، پاک کردن حافظه‌ی گرفته شده توسط آن برعهده‌ی کاربر است.
  - در قسمت‌های بعد، درباره‌ی انواع ویجت‌ها و چگونگی چیدن آن‌ها در صفحه صحبت خواهیم کرد.
- QDialog
  - کلاس پایه برای پنجره‌های دیالوگ است.
    - ❖ ویجت خاصی است که درون ویجت دیگری تعریف نمی‌شود (سطح اول برنامه‌ی کاربردی). اغلب برای وظایف کوتاه و ارتباطات کوتاه با کاربر به‌کار می‌رود.
  - دارای دو نوع modal و modeless است (مطالعه‌ی بیشتر از Qt Assistant). می‌تواند مقدار بازگشتی و دکمه‌های اولیه داشته باشد.

کلاس‌های پایه

- QMainWindow
  - پنجره‌ی اصلی برای یک برنامه را فراهم می‌کند. دارای چیدمان مخصوص به خود است.
    - می‌توان در آن چندین QToolBar و QDockWidgets، یک Qmenubar و یک QStatusBar افزود.
    - چیدمان دارای یک منطقه‌ی مرکزی است که می‌تواند توسط هر نوع ویجتی پر شود.
    - ساختن QMainWindow بدون ویجت مرکزی امکان‌پذیر نیست.
    - در MenuBar می‌توان Menu و برای Menuها Action تعریف نمود.



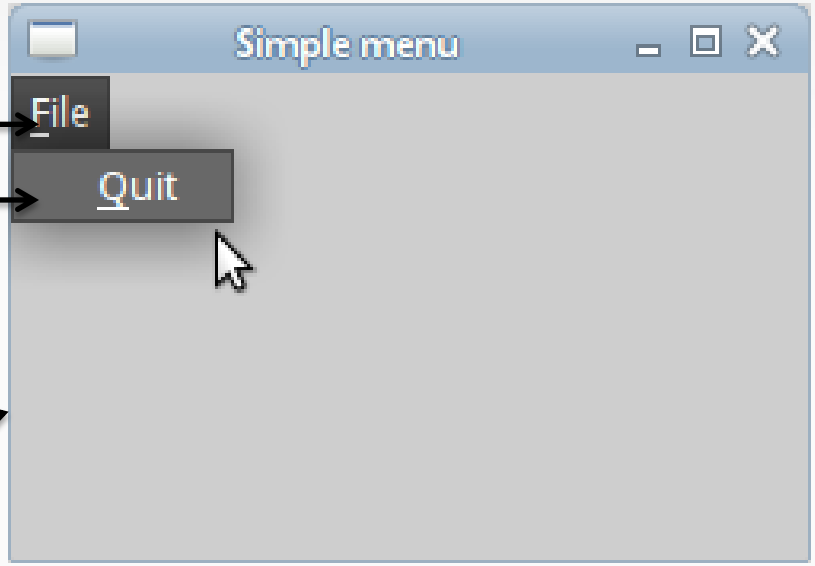
  - در ادامه مثالی ساده را بررسی خواهیم نمود.
    - فایل سرآیند
    - ارث‌بری از QMainWindow

```
//simplemenu.h
#include <QMainWindow>
#include <QApplication>
class SimpleMenu : public QMainWindow {
public:
        SimpleMenu(QWidget *parent = 0);
};
```

کلاس‌های پایه

- QMainWindow
  - فایل پیاده‌سازی simplewindow.cpp

```
//simplemenu.cpp
#include "simplemenu.h“
#include <QMenu>
#include <QMenuBar>
SimpleMenu::SimpleMenu(QWidget *parent)
        : QMainWindow(parent) {
                QAction *quit = new QAction("&Quit", this);
                QMenu *file;
                file = menuBar()->addMenu("&File");
                file->addAction(quit);
                connect(quit, SIGNAL(triggered()),
                        qApp, SLOT(quit()));
}
//main.cpp
#include "simplemenu.h“
int main(int argc, char *argv[]) {
                QApplication app(argc, argv);
                SimpleMenu window;
                window.resize(250, 150);
                window.move(300, 300);
                window.setWindowTitle("Simple menu");
                window.show();
                return app.exec();
}
```



کلاس‌های پایه

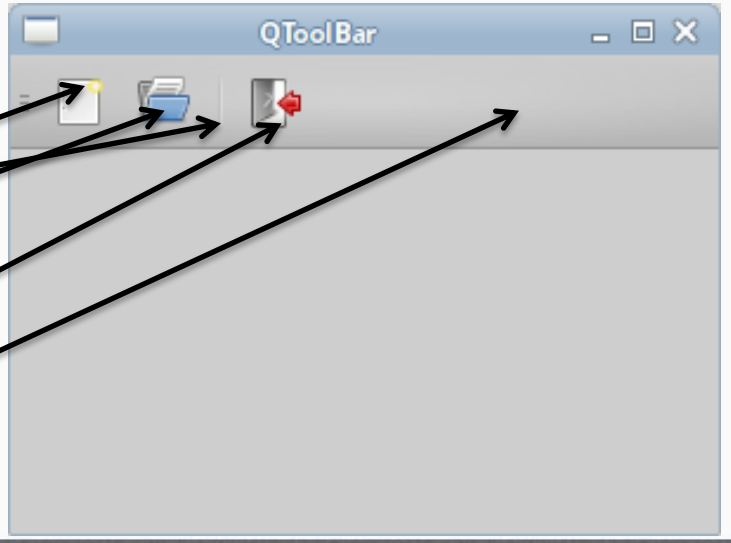
- QMainWindow

➢ افزودن عکس و کلید میان‌بر:

```
QPixmap quitpix("quit.png");
QAction *quit = new QAction(quitpix, "&Quit", this);
quit->setShortcut(tr("CTRL+Q"));
```

➢ افزودن نوار ابزار

```
toolbar->addAction(QIcon(newpix), "New File");
toolbar->addAction(QIcon(openpix), "Open File");
toolbar->addSeparator();
QAction *quit = toolbar->addAction(QIcon(quitpix),
                                   "Quit Application");
QToolBar *toolbar = addToolBar("main toolbar");
```



برخی کلاس‌های کاربردی

- QTextStream
  - جهت چاپ بر روی خروجی از یکی از موارد زیر استفاده می‌شود.

❖ کتابخانه‌ی استاندارد C

```
#include <iostream>
int main()
{
        std::cout << "console application\n";
}
```

❖ کتابخانه‌ی کیوت

- در محیط کیوت، استفاده از هر دو امکان‌پذیر است.

```
#include <QTextStream>
int main()
{
        QTextStream out(stdout);
        out << "console application\n";
}
```

برخی کلاس‌های کاربردی

S a régi szeretőmér
mit nem cselekednék,
tengerből a vizet
kanállal lemerném.

- QFile
  - متن روبه‌رو به زبان مجارستانی است.
    - ❖ فرض کنید در فایلی به نام szerelem ذخیره شده است.

```
#include <QTextStream>
#include <QFile>
int main() {
    QFile data("szerelem");
    QString line;
    if (data.open(QFile::ReadOnly)) {
        QTextStream in(&data);
        QTextStream out(stdout);
        out.setCodec("UTF-8");
        in.setCodec("UTF-8");
        do {
            line = in.readLine();
            out << line << endl;
        } while (!line.isNull());
    }
}
```

  - برنامه‌ی زیر، این متن را از فایل szerelem خوانده و بر روی مانیتور نمایش می‌دهد.
    - ❖ توضیح درباره‌ی QString در اسلایدهای بعد

- همان‌طور که مشاهده شد، با استفاده از QFile و با کمک QTextStream می‌توان بر روی فایل نوشت یا از فایل خواند.

برخی کلاس‌های کاربردی

- QList
  - ابزاری مناسب برای کار با لیست‌ها (لیستی از نمونه‌ها) فراهم آورده است.
  - برای مثال، در کد زیر، لیستی از اسامی ساخته، آنها را مرتب کرده و نتیجه را در خروجی چاپ می‌نماییم.

```
#include <QTextStream>
#include <QList>
int main() {
        QTextStream out(stdout);
        QList<QString> list;
        list << "Balzac" << "Tolstoy" <<
        "Guldbrassen" << "London" <<
        "Galsworthy" << "Sienkiewicz";
        qSort(list);
        for (int i = 0; i < list.size(); ++i) {
                out << list.at(i) << endl;
        }
}
```

تعریف لیست

افزودن مقادیر به لیست

مرتب کردن لیست

چاپ لیست در خروجی استاندارد

برخی کلاس‌های کاربردی

- QDir
  - جهت دسترسی به دایرکتوری‌های سیستم عامل و محتوای آنها استفاده می‌شود.
  - مثال:
    - در برنامه‌ی زیر، کلیه‌ی فایل‌ها در پوشه‌ی جاری را پیدا کرده و یک فیلتر خاص بر روی آن‌ها اعمال می‌کنیم.

```
#include <QTextStream>
#include <QDir>
int main() {
    QTextStream out(stdout);
    QDir dir;
    QStringList filters;
    filters << "*.c" << "*.c~";
    dir.setNameFilters(filters);
    QFileInfoList list = dir.entryInfoList();
    for (int i = 0; i < list.size(); ++i) {
        QFileInfo fileInfo = list.at(i);
        out << QString("%1").arg(fileInfo.fileName());
        out << endl;
    }
}
```

برخی کلاس‌های کاربردی

- QImage
  - جهت پردازش و کار بر روی تصاویر می‌توان از آن بهره گرفت.
  - مهم‌ترین توابع آن به شرح زیر است:
    - بازکردن عکس
      - QImage img(“path/filename”);
    - ذخیره‌سازی عکس
      - img.save(“path/filename”);
    - خواندن یک نقطه
      - QRgb colors = img.pixel(i,j);
    - نوشتن یک نقطه
      - img.setPixel(i,j,colors);

برخی کلاس‌های کاربردی

- QTime
  - توابع کار با ساعت سیستم
  - مثال
    - کد زیر زمان جاری سیستم را بر روی خروجی نمایش می‌دهد.

```
#include <QTextStream>
#include <QTime>
int main() {
        QTextStream out(stdout);
        QTime qtime = QTime::currentTime();
        QString stime = qtime.toString(Qt::LocalDate);
        out << stime << endl;
}
```

برخی کلاس‌های کاربردی

- QString

➢ الحاق سه رشته

```
#include <QTextStream>
int main() {
        QString a = "Disziplin ";
        QString b = "ist ";
        QString c = "Macht.\n";
        QTextStream out(stdout);
        out << a + b + c;
}
```

➢ افزودن به انتهای رشته

```
#include <QTextStream>
int main() {
        QString string = "Whether I shall ";
        string.append("turn out to be the ");
        string.append(" hero of my own life, \n ");
        string.append("these pages must
show.\n");
        QTextStream out(stdout);
        out << string;
}
```

➢ جابه‌جایی آرگومان با مقدار

```
#include <QTextStream>
int main() {
        QString string = "What if I gave you %1 red
roses?";
        int num = 21;
        QTextStream out(stdout);
        out << string.arg(num) << endl;
}
```

➢ تبدیل رشته به حروف بزرگ یا کوچک

```
#include <QTextStream>
int main() {
        QString string = "The history of my life.";
        QTextStream out(stdout);
        out << string.toLower() << endl;
        out << string.toUpper() << endl;
}
```

برخی کلاس‌های کاربردی

- QString
  - شمارش تعداد کاراکترهای موجود در رشته.

```
#include <QTextStream>
int main() {
        QString string = "The history of my life.";
        QTextStream out(stdout);
        out << "The string has " + String::number(string.size())
                                  + " characters.“
            << endl;
}
```

- برای مطالعه‌ی بیشتر بر روی کلاس‌های QString و QTime به ترتیب به بخش‌های ۳ و ۴ از منبع [8] مراجعه نمایید.

برخی کلاس‌های کاربردی

- QPainter
  - برای رسم توسط کیوت، دو روش عمده وجود دارد:
    - استفاده از OpenGL (در بخش دو آن را بررسی خواهیم نمود).
    - استفاده از QPainter و کلاس‌های مرتبط با آن
  - در اینجا تنها QPainter را معرفی خواهیم نمود.
    - رسم توسط کلاس QPainter پس از رخ دادن رویداد paintEvent() صورت خواهد پذیرفت.
      - در ادامه، راجع به رویدادها صحبت خواهیم کرد.
  - نکته: از QPainter تنها برای ترسیمات دوبعدی استفاده می‌شود.
  - برای مطالعه‌ی بیشتر در این زمینه، به منابع زیر مراجعه نمایید:
    - منبع [1] فصل ۸
    - منبع [8] بخش ۱۱

برخی کلاس‌های کاربردی

- QPainter

➢ مثال: رسم خط در صفحه

❖ رویداد رسم هنگامی فراخوانی می‌شود که یک ویجت به‌روز شود.

❖ کلاس QPen برای رسم خطوط و خطوط بیرونی شکل کاربرد دارد.

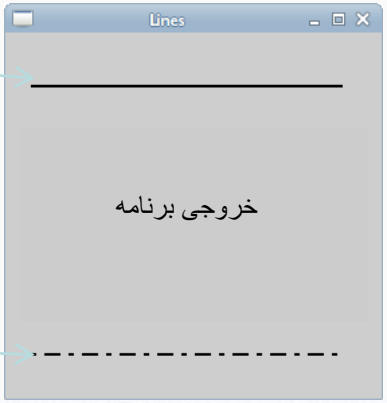
❖ برای Pen می‌توان سبکی جدید انتخاب نمود.

❖ برای رسم خط از drawLine استفاده می‌شود.

– پارامترها مختصات دو راس پاره‌خط هستند.

```
class Lines : public QWidget {
        Q_OBJECT
protected:
        void paintEvent(QPaintEvent *event);
};
```

```
#include <QPainter>
void Lines::paintEvent(QPaintEvent *e) {
        QPainter qp(this);
        QPen pen(Qt::black, 2, Qt::SolidLine);
                qp->setPen(pen);
        qp->drawLine(20, 40, 250, 40);
                QVector<qreal> dashes;
                qreal space = 4;
                dashes << 1 << space << 5 << space;
                pen.setStyle(Qt::CustomDashLine);
                pen.setDashPattern(dashes);
                qp->setPen(pen);
        qp->drawLine(20, 240, 250, 240);
}
```



برخی کلاس‌های کاربردی

- QPainter
  - رسم مستطیل با رنگ و الگوی دلخواه

```
QPainter painter(this);
painter.setPen(QColor("#d4d4d4"));
painter.setBrush(QBrush("#c56c00"));
painter.drawRect(10, 15, 90, 60);
```

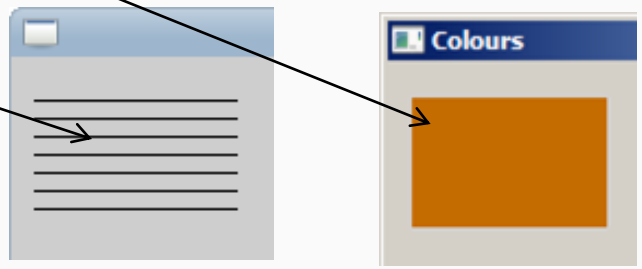
SetPen() برای تنظیم خطوط اطراف مناسب است

setBrush() رنگ درون شکل را تنظیم میکند (fill pattern)

رسم مستطیل مورد نظر

```
painter.setBrush(Qt::HorPattern);
```

HorPattern یک ثابت برای ساختن الگوی خطوط موازی است




```
painter.setRenderHint(QPainter::Antialiasing);
```

رسم در مد Antialiasing (کیفیت بالای Rendering) انجام خواهد گرفت

```
painter.rotate(5.0);
```

چرخش: به اندازه‌ی ۵ درجه

```
painter.translate(QPoint(width()/2, height()/2));
```

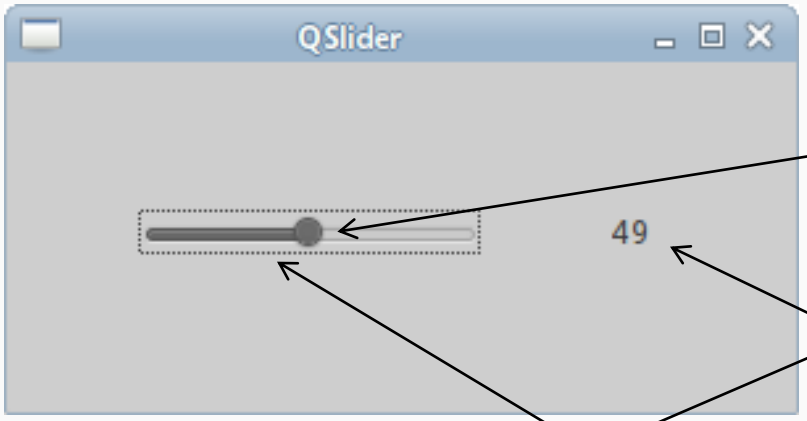
انتقال: شروع رسم از وسط پنجره و نه مبدا (۰،۰)

برخی از ویجت‌ها

- QLabel
  - نمایش متن یا عکس با تنظیمات برنامه‌نویس
    - قابل دستکاری توسط کاربر نیست.
    - مثال: تنظیم فونت متن ویجت توسط برنامه‌نویس:

```
label->setFont(QFont("Purisa", 10));
```

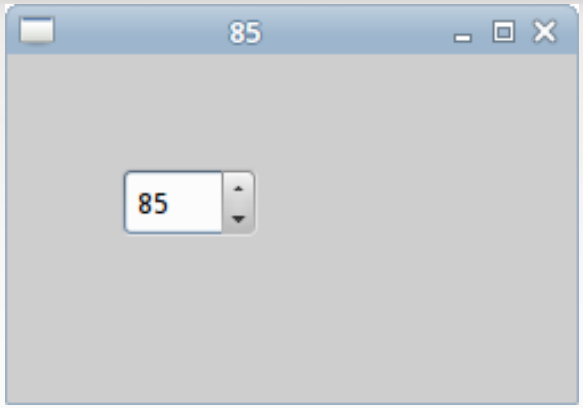
- QSlider
  - انتخاب مقدار دلخواه توسط کاربر
    - مثال: متصل ساختن
      - سیگنال تغییر مقدار slider به
      - اسلات نوشته شده برای افزایش عدد




```
connect(slider, SIGNAL(valueChanged(int)), label, SLOT(setNum(int)));
```

  - با دستور connect در ادامه بیشتر آشنا خواهیم شد.

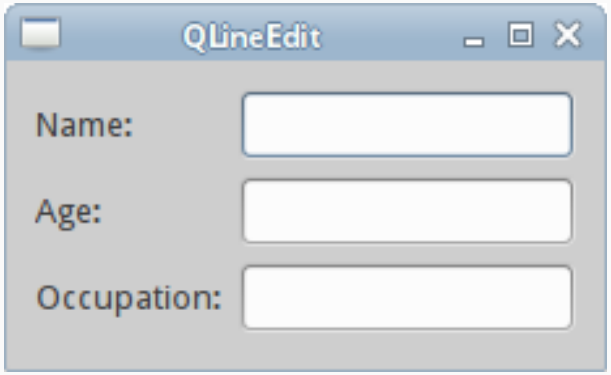
برخی از ویجت‌ها



- QSpinBox
  - برای رسیدگی به مقادیر صحیح و گسسته
    - مثال: مقدار انتخاب شده در ویجت بر روی عنوان پنجره نمایش داده می‌شود.

```
connect(spinbox, SIGNAL(valueChanged(int)), this, SLOT(setTitle(int)));
```

- QLineEdit
  - وارد کردن یا تغییر یک خط متن توسط کاربر
  - دارای توابع undo/redo, cut/paste and drag & drop
  - مثال:
    - فرم روبه‌رو از سه QLabel و سه QLineEdit متناظر با آنها تشکیل شده است.
    - کاربر قادر به نوشتن بر روی QLabelها نیست اما می‌تواند در QLineEditها بنویسد.



برخی از ویجت‌ها

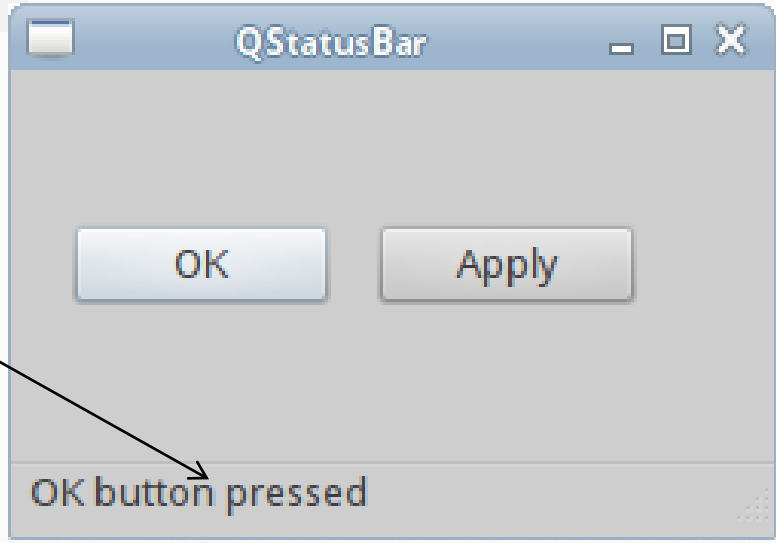
- QStatusBar
  - برای نمایش اطلاعات وضعیتی برنامه کاربرد دارد.
    - برای به نمایش درآمدن نوار وضعیت بایستی statusBar() را فراخوانی کرد.
    - تابع showMessage، پیام را بر روی نوار وضعیت نشان می‌دهد.

```
statusBar();
```

```
connect(ok, SIGNAL(clicked()), this, SLOT(OnOkPressed()));
connect(apply, SIGNAL(clicked()), this, SLOT(OnApplyPressed()));
```

```
void Statusbar::OnOkPressed() {
	statusBar()->showMessage("OK button pressed", 2000);
}
```

```
void Statusbar::OnApplyPressed() {
	statusBar()->showMessage("Apply button pressed", 2000);
}
```

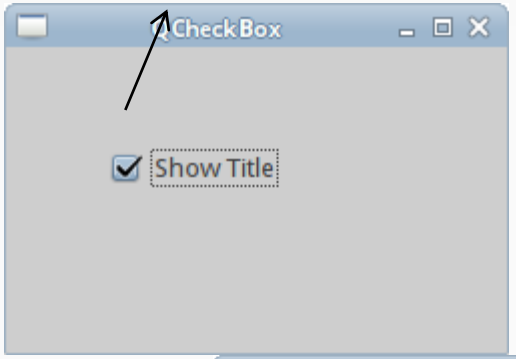


برخی از ویجت‌ها

```
void Class::showTitle(int state) {
        if (state == Qt::Checked){
                setWindowTitle("QCheckBox");
        } else {
                setWindowTitle("");
        }
}
```

- QCheckBox
  - تشکیل‌شده از یک box و یک label است.

```
connect(cb, SIGNAL(stateChanged(int)), this, SLOT(showTitle(int)));
```



- QListWidget
  - نمایش لیستی از آیتم‌ها
    - در مثال زیر نام فیلم‌ها آیتم هستند.


```
QListWidget *lw;
```

```
lw->addItem(r);
```
افزودن آیتم به لیست

```
lw->takeItem(r);
```
حذف آیتم از لیست

```
lw->currentItem();
```
آیتمی که اکنون انتخاب شده است

برخی از ویجت‌ها

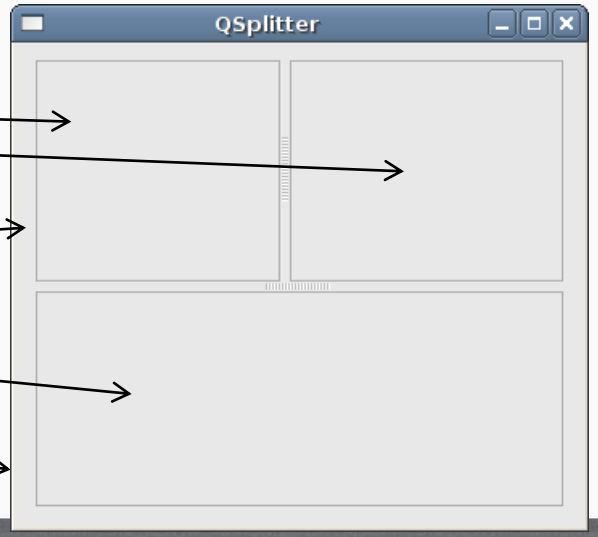
- QPixmap
  - برای نمایش عکس بر روی صفحه بهینه‌سازی شده است.
    - ساختن یک pixmap و افزودن آن به درون یک ویجت label :

```
QPixmap pixmap("bojnice.jpg");
```

```
QLabel *label = new QLabel(this);
label->setPixmap(pixmap);
```

- QSplitter
  - به کاربر اجازه‌ی کنترل اندازه‌ی ویجت‌های فرزند را می‌دهد.

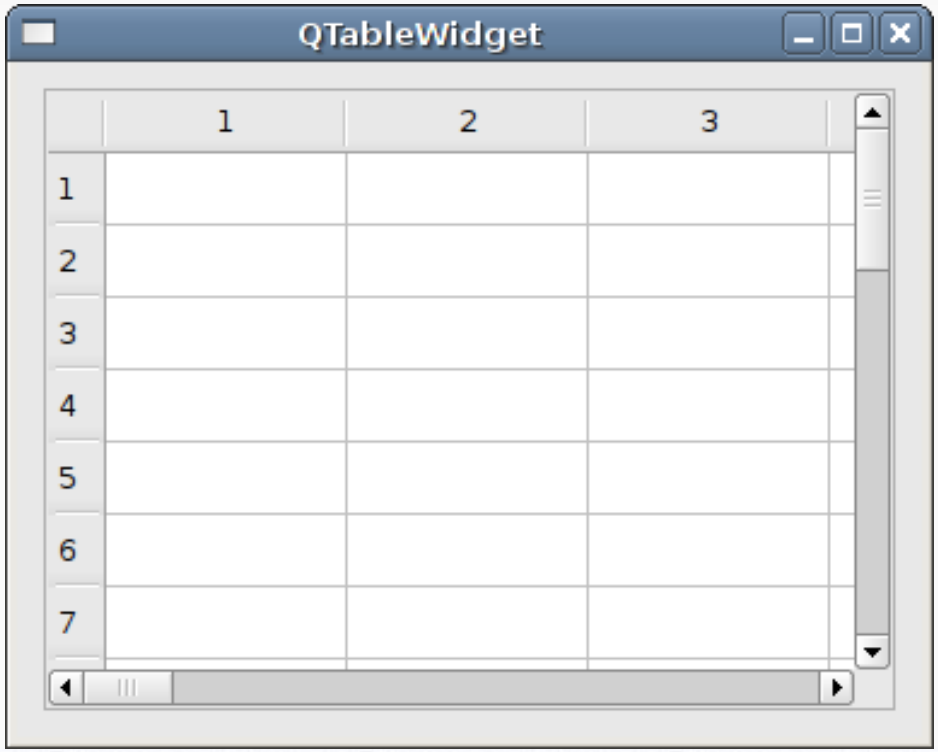
```
QSplitter *splitter1 = new QSplitter(Qt::Horizontal, this);
splitter1->addWidget(topleft);
splitter1->addWidget(topright);

QSplitter *splitter2 = new QSplitter(Qt::Vertical, this);
splitter2->addWidget(splitter1);
splitter2->addWidget(bottom);
QHBoxLayout *hbox = new QHBoxLayout(this);
hbox->addWidget(splitter2);
setLayout(hbox);
```



برخی از ویجت‌ها

- QTableWidget (grid widget)
  - برای ساختن صفحات گسترده (مثلاً Excell) مفید است.

```
QTableWidget *table = new QTableWidget(25, 25, this);
QHBoxLayout *hbox = new QHBoxLayout(this);
hbox->addWidget(table);
setLayout(hbox);
```



برخی از ویجت‌ها

- ایجاد ویجت جدید
  - در صورتی که ویجت مورد نیاز ما وجود نداشته باشد چاره چیست؟
  - راه حل: برنامه‌نویس بایستی بتواند ویجت‌های مورد نیاز خود را بسازد.
    - با استفاده از ابزارهای رسم فراهم شده توسط جعبه‌ابزار
  - روش اول
    - ایجاد تغییر در یکی از ویجت‌های موجود (Modify).
    - بهبود بخشیدن به یکی از ویجت‌های موجود (Enhance).
    - ترکیب دو یا چند ویجت موجود و ساخت یک ویجت جدید.
    - ترفیع دادن یکی از ویجت‌های موجود (Promote).
  - روش دوم
    - ساختن یک ویجت کاملاً جدید با استفاده از ابزار رسم (Qpainter یا OpenGL)

چیدمان واسط کاربری

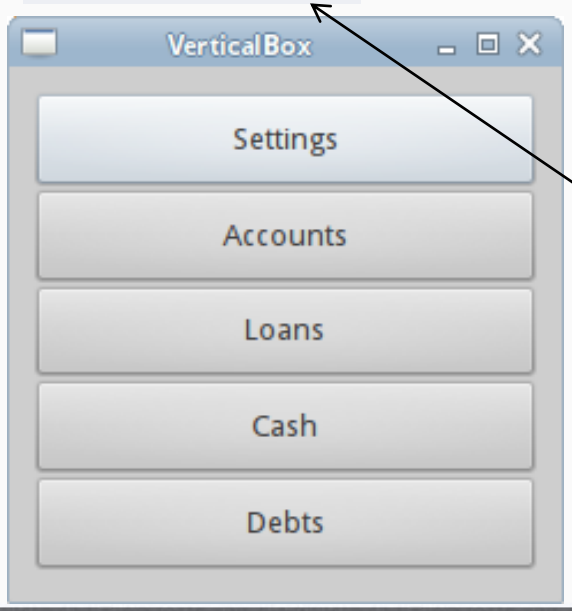
- یکی از نیازهای برنامه‌نویس gui تنظیم مکان ویجت‌ها بر روی پنجره به‌گونه‌ای است که پنجره ظاهری مناسب داشته باشد.
  - کیوت این امکان را در اختیار برنامه‌نویس قرار داده تا ویجت‌ها را به روش دلخواه خود در پنجره نمایش دهد.
- برای چیدمان صفحه، دو راه داریم:
  - چیدمان قطعی: برنامه‌نویس مکان و اندازه‌ی هر ویجت را برحسب پیکسل مشخص می‌کند.
  - QLayout: برنامه‌نویس برای چیدمان صفحه از ابزارهای layout کیوت کمک می‌گیرد.
- معایب چیدمان قطعی:
  - ❖ با تغییر ابعاد پنجره، سایز و مکان ویجت‌ها تغییری نمی‌کند.
  - ❖ برنامه‌ی تولیدی در پلت‌فرم‌های مختلف، ظاهری متفاوت خواهد داشت.
  - ❖ تغییر در فونت باعث به هم ریختن layout برنامه می‌شود.
  - ❖ در صورتی که بخواهید layout را تغییر دهید، بایستی تمامی مراحل را از ابتدا انجام دهید.

مثالی از چیدمان قطعی

```
QTextEdit *edit = new QTextEdit(this);
edit->setGeometry(5, 5, 200, 150);
```

چیدمان واسط کاربری

```
QVBoxLayout *vbox = new QVBoxLayout(this);
vbox->setSpacing(1);
vbox->addWidget(settings);
vbox->addWidget(accounts);
vbox->addWidget(loans);
vbox->addWidget(cash);
vbox->addWidget(debts);
setLayout(vbox);
```



- QLayout
  - ویجت‌ها را می‌توان در layoutهای افقی، عمودی، گرید و یا ترکیبی از layoutها چید.
  - اندازه‌ی واقعی (size) توسط size policy هر ویجت تعیین می‌شود.
- QVBoxLayout
  - ویجت‌ها را به صورت عمودی می‌چیند.
  - مثال:
    - یک layout عمودی می‌سازیم.
    - فاصله‌ی بین ویجت‌های فرزند = ۱ پیکسل
    - ویجت‌های مورد نظر را در layout اضافه می‌کنیم.
    - مدیر layout پنجره‌ی فعلی، vbox است.

چیدمان واسط کاربری

- QHBoxLayout
  - ویجت‌ها را به صورت افقی می‌چیند.
  - تنظیمات، مشابه با QVBoxLayout
- برخی تنظیمات دیگر
  - افزودن ویجت با مشخصات خاص ← `hbox->addWidget(ok, 1, Qt::AlignRight);`
  - افزودن یک فضای خالی و قابل افزایش ← `vbox->addStretch(1);`
  - ساخت layoutهای تودرتو ← `vbox->addLayout(hbox);`
- QGridLayout
  - ویجت‌ها را به فرم گرید می‌چیند.
- برای مطالعه‌ی بیشتر در زمینه‌ی چیدمان صفحه:
  - منبع [1] فصل ۶

Events,Signals,Slots

- اکثر برنامه‌های کاربردی GUI مبتنی بر رویداد هستند.
  - برنامه، نسبت به رویدادهای مختلف حین اجرا، واکنش نشان می‌دهد.
  - رویدادها عموماً توسط کاربر، اتصال اینترنت، مدیر پنجره و تایمر ایجاد می‌شوند.
- برای یک رویداد، سه عضو اصلی مدل می‌شوند:
  - منبع رویداد (Event Source)
    - ❖ شئ‌ای است که وضعیت آن تغییر می‌کند و رویدادی تولید می‌کند.
  - شئ رویداد (Event Object)
    - ❖ تغییرات وضعیت در منبع رویداد را دسته‌بندی و مجزا می‌کند.
  - هدف رویداد (Event Target)
    - ❖ شئ‌ای است که می‌خواهد از وقوع رویداد باخبر شود به وظیفه‌ی رسیدگی به رویداد را بر عهده دارد.
- در کیوت برای رسیدگی به رویداد (Event Handling) دو راه وجود دارد:
  - استفاده از مکانیزم سیگنال-اسلات.
  - بازنویسی توابع رسیدگی به رویداد

Events,Signals,Slots

- سیگنال
  - یک QObject می‌تواند در هر زمان در صورت نیاز اقدام به انتشار (emit) سیگنال‌های مناسب نماید.
  - با این وجود اطلاعی از این‌که چه کسی به سیگنال گوش خواهد داد ندارد.
- اسلات
  - هر QObject می‌تواند دارای تعدادی اسلات باشد.
  - اسلات یک متد معمولی C++ است.
  - اسلات‌ها می‌توانند در زمان اجرا به سیگنالی متصل شده یا از آن جدا شوند.
  - این اتصالات بدون نیاز به اطلاع‌یافتن اشیای درون برنامه صورت می‌پذیرد.
- نکته: در صورتی که یک کلاس قصد استفاده از سیگنال‌ها و اسلات‌ها یا سایر خدمات ابزار moc از کیوت را دارد بایستی ماکروی Q_OBJECT را در قسمت private تعریف خود نوشته باشد.
- برای برقراری یا قطع ارتباط بین یک سیگنال و اسلات از دستورات زیر استفاده می‌شود.
  - برقراری ارتباط:

```
connect( *obj1, SIGNAL(signame(varType)), *obj2, SLOT(sloname(varType)) );
```

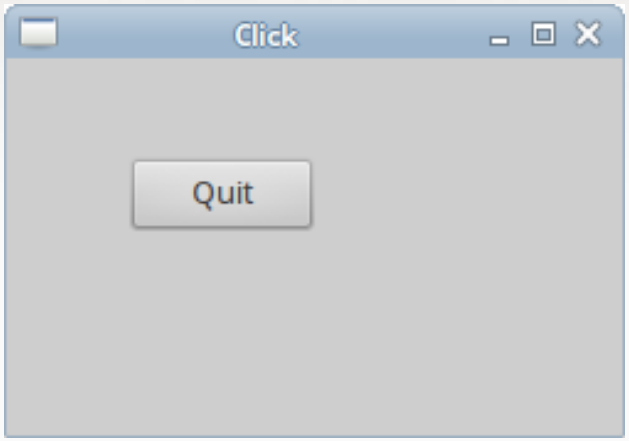
  - قطع ارتباط:

```
disconnect( *obj1, SIGNAL(signame(varType)), *obj2, SLOT(sloname(varType)) );
```

Events,Signals,Slots

- مثال

```
connect( quit, SIGNAL(clicked()), qApp, SLOT(quit()) );
```



- سیگنال clicked() از شئ quit (اشاره‌گری به QPushButton)
- اسلات quit() از شئ qApp (یک اشاره‌گر سراسری به برنامه)
- با دستور connect این سیگنال و اسلات را متصل می‌کنیم.
- **نتیجه:** کلیک بر روی دکمه‌ی quit سبب بسته شدن پنجره می‌شود.

- مثال
  - در قطعه‌کد سمت چپ، به نحوه‌ی تعریف سیگنال‌ها و اسلات‌ها توجه نمایید.
  - در قطعه کد سمت راست، مثالی از اتصال و قطع اتصال آمده است.

```
class Disconnect : public QWidget {
    Q_OBJECT
        Disconnect(QWidget *parent = 0);
private slots:
        void onCheck(      int);
private signals:
         void clicked();
private:
          QPushButton *click;
};
```

```
void Disconnect::onCheck(int state) {
      if (state == Qt::Checked) {
            connect(click, SIGNAL(clicked()), this, SLOT(onClick()));
      } else {
            click->disconnect(SIGNAL(clicked()));
      }
}
```

Events,Signals,Slots

```
//eventhandler.h
#include <QWidget>
class EveHand : public QWidget {
protected:
        void keyPressEvent(QKeyEvent * e);
        void moveEvent(QMoveEvent *e);
};
```

```
#include <QApplication>
#include <QKeyEvent>
#include <QMoveEvent>
#include “eventhandler.h“
void EvetHand::keyPressEvent(QKeyEvent *event) {
        if (event->key() == Qt::Key_Escape) {
                qApp->quit();
        }
}
void EveHand::moveEvent(QMoveEvent *e) {
        int x = e->pos().x();
        int y = e->pos().y();
        QString text = QString::number(x) + "," +
        QString::number(y);
        setWindowTitle(text);
}
```

- بازنویسی توابع رسیدگی به رویداد
  - مثال
  - بازنویسی مجدد رویدادهای:
    - ❖ فشرده شدن کلیدی از صفحه‌کلید
      - با فشرده‌شدن کلید Esc توسط کاربر از برنامه خارج خواهیم شد.
    - ❖ جابه‌جا شدن ویجت
    - ❖ رویداد تایمر:

```
protected:
        void timerEvent(QTimerEvent *e);
```

```
void Class::timerEvent(QTimerEvent *e) {
        Q_UNUSED(e);
        QTime qtime = QTime::currentTime();
        QString stime = qtime.toString();
        label->setText(stime);
}
```

منابع مفید برای مطالعه

- وب‌سایت‌ها
  - سایت رسمی کیوت:
    - http://qt-project.org/
  - لیست وبلاگ‌های مفید:
    - http://planet.qt-project.org/
  - سایت رسمی (تجاری) کیوت
    - http://qt.digia.com/
  - لیستی از مثالهای مناسب:
    - http://qt-project.org/doc/qt-5.0/qtdoc/qtexamplesandtutorials.html
- Qt Documentations
  - یکی از منابع غنی برای آموزش کیوت، مستندات خود این ابزار می‌باشد که به صورت زیر در دسترس است:
    - مستندات به صورت آنلاین: http://qt-project.org/doc/qt-5.1/qtdoc/index.html
    - از طریق ابزار Qt Assistant

منابع مفید برای مطالعه

- کتاب‌ها
  - برای دسترسی به این کتاب‌ها به فولدر روبه‌رو مراجعه نمایید:

Qt\Books

- [1] C++ GUI Programming With QT4(2nd Edition)
- [2] Advanced Qt Programming
- [3] An Introduction to Design Patterns in CPP with Qt 2nd Edition
- [4] Qt Cross-platform C++ GUI Application Framework Technical Overview
- [5] Qt Quick Game Programming 1_0
- [6] Introduction to Qt4
- [7] Qt_by_Mehrdad_Momeni
- [8] Qt4 tutorial

منابع مفید برای مطالعه

- نمونه کدها
  - برای مشاهده و اجرای برخی نمونه کدها به فولدر زیر مراجعه نمایید.

```
Qt\Codes
```

- فایل‌های نصب
  - فایل‌های نصب کیوت (لینوکس و ویندوز)، در فولدر زیر یافت می‌شوند.

```
Qt\Setup
```

- فایل‌های آموزشی
  - مجموعه‌ای از فایل‌های آموزشی و کاربردی (بعضاً حاوی کدهای قابل اجرا) در فولدر زیر قرار دارند.

```
Qt\Tutorials
```