Практика 2.

# Cобытия мыши.Tooltips.Начинаем рисовать. Класс QPainter. События изменения размеров окна. События таймера.

## События

### События мыши.

### Обработчик нажатия мыши

Для обработки события о нажатии правой кнопки мыши в нашем окне перегружаем виртуальный метод:

voidQWidget::mousePressEvent ( [QMouseEvent](D:\\C++\\QT\\VORONEG\\PRACTIKA 2\\qmouseevent.html) \* *event* ) [virtual protected]

Проверьте, что при **любом** нажатии на **любую** кнопку мыши вызывается Ваша заглушка

### При нажатии правой кнопки выводим текущие координаты курсора с помощью всплывающей подсказки –tooltip:

Все обработчики событий мыши принимают в качестве параметра указатель на объект [QMouseEvent](file:///D:\Marina\Qt\qmouseevent.html) (следует подключить заголовочный файл <QMouseEvent>), в котором запакована следующая информация:

* Какая из кнопок мыши нажата. Для получения этой информации можно использоватьметод  
  Qt::MouseButtonsQMouseEvent::buttons()  
  , который возвращает значение, соответствующее одной из (или комбинации) именованных констант: Qt::NoButton, Qt::LeftButton, Qt::RightButton, Qt::MidButton…)
* Координаты курсора мыши относительно левого верхнего угла окна (клиентские координаты). Методы  
  int QMouseEvent::x()const  
  и int QMouseEvent::y()const  
  или const QPoint&QMouseEvent::pos () const
* Координаты курсора мыши относительно левого верхнего угла экрана. Методы  
  int QMouseEvent::globalX()const  
  и int QMouseEvent::globalY()const  
  или constQPoint&QMouseEvent::globalPos () const
* Событие происходит при нажатой системной клавише. Метод  
  Qt::KeyboardModifiersQKeyEvent::modifiers () const  
  устанавливает биты в соответствии с нажатыми системными клавишами. Для проверки возвращаемого значения используются именованные константы: Qt::ShiftModifier, Qt::ControlModifier, Qt::ControlModifier…

**Задание 1.1**

В строке подсказки (tooltip) выведите текущие координаты курсора мыши, полученные при нажатии правой кнопки мыши (находящейся над виджетом).

Порядок действий:

1. В обработчике mousePressEvent() анализируем, какая кнопка была нажата (нас интересует правая). Для этого вызываем метод QMouseEvent::button() или QMouseEvent::buttons()
2. Если нажата правая кнопка, из полученного параметра извлекаем ЭКРАННЫЕ координаты мыши (метод pos() возвращает относительные, метод globalPos() – абсолютные)
3. Формируем строку подсказки вида «x=<значение>, y=<значение>». В эту строчку логичнее выводить относительные (клиентские) координаты.  
   Подсказки:
   1. для формирования результирующей строки для вывода рекомендуется пользоваться QTextStream:  
      QString str;  
      QTextStream tstream(&str);  
      tstream<<"x="<<…
   2. в новых версиях Qt не рекомендуется использовать QString::sprintf()
   3. для конкатенации строк можно использовать перегруженные операторы “+” и “+=”класса QString
   4. для перевода целого значения в строковое представление можно пользоваться методом QString::setNum()
4. Не забудьте #include<QToolTip>
5. Вывод подсказки осуществляется методами:

void QToolTip::showText ( const [QPoint](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qpoint.html) & *pos*,const [QString](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qstring.html) & *text*, [QWidget](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html) \* *w=0* ) [static]  
и  
voidQToolTip::showText ( const [QPoint](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qpoint.html) & *pos*, const [QString](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qstring.html) & *text*, [QWidget](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qwidget.html) \* *w*, const [QRect](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qrect.html)& *rect* ) [static]  
, где pos должна быть задана в экранных координатах

### А теперь будем выводить tooltip при нажатии правой кнопки мыши, если только нажата клавиша Ctrl

Подсказка: для проверки используем метод QMouseEvent::modifiers()

**Задание 1.2**

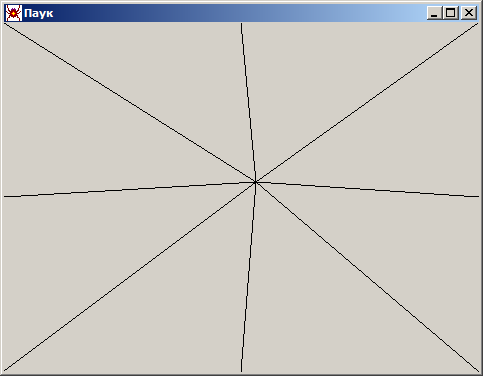
Выведите в tooltip координаты мыши, полученные при нажатии правой кнопки мыши, только в том случае, если нажата клавиша-модификатор Ctrl.

### Рисуем паутину при нажатой левой кнопке мыши

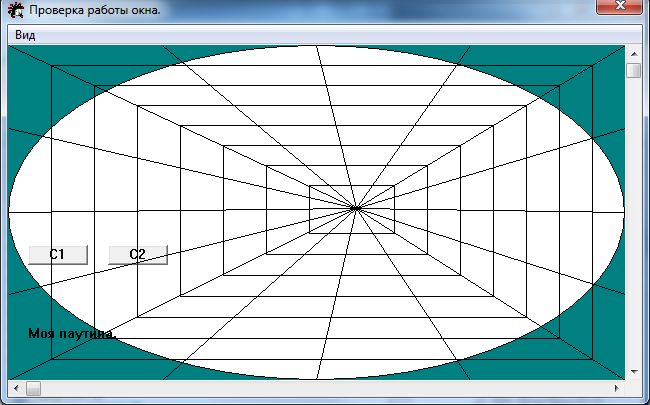
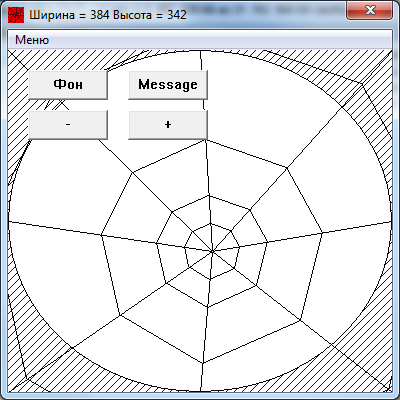
**Задание 2**

Нарисуйте на виджете паутину. Особенность паутины в том, что центр ее должен находиться в позиции курсора, а вторые концы отрезков паутины могут располагаться по краям прямоугольника рабочей области. Причем при изменении позиции курсора паутина должна перерисовываться.

Самая примитивная паутинка может выглядеть так:



Развивать эту тему можно бесконечно в зависимости от Ваших математических и эстетических возможностей.

Порядок действий:

1. Так как паутину нужно рисовать далеко не всегда (а только при нажатой левой кнопке мыши)=> нам нужен признак =>в классе Spider объявите булеву переменную, например, isDrawing. Как и где нужно проинициализировать эту переменную?
2. При нажатии левой кнопки мыши (в обработчике mousePressEvent()) флажок isDrawing нужно устанавливать
3. При отпускании соответственно сбрасывать =>перегрузите виртуальный метод  
   void QWidget::mouseReleaseEvent([QMouseEvent](qthelp://org.qt-project.qtwidgets.502/qtgui/qmouseevent.html) \**event*) [virtual protected]
4. При перемещении мыши  
   void QWidget:: mouseMoveEvent ([QMouseEvent](file:///C:\Users\user\AppData\Roaming\Microsoft\Word\qmouseevent.html) \* *event* ) [virtual protected]  
   если требуется рисовать, вызывать перерисовку окна посредством метода  
   void QWidget::repaint ()

Замечания:

* по умолчанию виртуальный метод mouseMoveEvent вызывается только в том случае, когда пользователь двигает мышь при нажатой кнопке
* Собственно рисование паутины будет осуществляться в перегруженном обработчике события перерисовки окна - paintEvent()
* !!! Обратите внимание!!! Qt «захватывает мышь», если нажимается любая кнопка мыши. Это означает, что все события мыши (события о перемещении мыши) будут направляться окну (независимо от того, вышел курсор за пределы окна или нет). Мышь будет «захвачена» до того момента, пока не будет отжата последняя кнопка. И даже вид курсора меняться не будет!

**Важное замечание:**

### [Qt::MouseButton](qthelp://org.qt-project.qtcore.551/qtcore/qt.html#MouseButton-enum) QMouseEvent::button() const – возвращает кнопку, вызвавшую событие. Для события mouseMoveEvent всегда возвращается значение [Qt::NoButton](qthelp://org.qt-project.qtcore.551/qtcore/qt.html#MouseButton-enum).

## Начинаем рисовать

### Перегрузите виртуальный метод QWidget::paintEvent()

void QWidget::paintEvent ( [QPaintEvent](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qpaintevent.html) \* *event* ) [virtual protected]

Замечание:  
Обработчик paintEvent() **никогда не вызывается явно**. Перерисовку может инициировать сама ОС (например, когда требуется восстановить окно, временно перекрытое другим окном…) или опосредованно программист может инициировать перерисовку, вызвав метод  
void QWidget::**repaint** () [slot] – вызывает немедленную перерисовку  
или void QWidget::**update** () [slot]– поставить событие о перерисовке в очередь => оптимизации =>может обработать несколько событий о перерисовке за один вызов paintEvent()

# Прежде, чем собственно что-либо нарисовать

Для рисования нужно подготовиться:

* создать объект, которому Вы будете делегировать все операции вывода.

Класс QPainter инкапсулирует системный контекст устройства и осуществляет вывод текста, векторной и растровой графики на любое стандартное устройство вывода (дисплей, принтер, плоттер…), а также содержит средства для масштабирования, задания системы координат, хранит инструменты для рисования…

Замечание:

для подавления «мерцания» при перерисовке в Qt используется двойная буферизация (изображение формируется в памяти, а потом переносится на экран).

#include<QPainter>

При создании объекта QPainter в конструктор в качестве параметра передается QPaintDevice\* - указатель на соответствующий контекст устройства (это объект, который содержит информацию о том, куда мы собираемся осуществлять вывод: в окно, на принтер…).

* Обычно объекты QPainter создаются в **обработчике события о перерисовке**. Там же происходит рисование.

# Рисование линий

Напоминание: рисовать стоит только в том случае, когда установлен флаг IsDrawing

* Создаем объект QPainter. В качестве параметра конструктора указываем наше окно
* Получаем размеры клиентской области  
  QRect QWidget::rect() const
* Получаем текущее положение курсора

1 способ

Можно получить текущее положение курсора в событии QPaintEvent (**в координатах экрана**) посредством статического метода класса QCursor  
QPoint QCursor::pos() [static]

А рисовать будем **в клиентских координатах** =>перевести из глобальных в оконные можно с помощью:  
QPoint QWidget::mapFromGlobal(constQPoint&pos) const

2 способ

А можно завести в классе Spider переменную типа QPoint и фиксировать текущее положение курсора в обработчиках события QMouseEvent: mousePressEvent и mouseMoveEvent.

В качестве параметра в этих обработчиках используется QMouseEvent\* event. Можно получить текущуюю позицию курсора с помщью метода:

QPoint QMouseEvent::pos() const;

* Рисуем паутину, используя текущую точку курсора и размеры клиентской области
* Подсказка: для рисования отрезков можно использовать методы  
  QPainter::drawLine()  
  QPainter::drawPolyline()  
  или QPainter::drawLines()
* Подсказка: у класса QRect есть полезные методы для получения левого верхнего угла (topLeft()), …
* Цвет паутины тоже можно задавать по желанию

## Курсор

По умолчанию курсор имеет форму стрелки. Изменить вид курсора можно посредством:  
void QWidget::**setCursor** ( constQCursor& );

восстановить умолчание – посредством:  
voidQWidget::**unsetCursor** ();

Замечание:   
курсор принимает заданную форму только в пределах **клиентской** области того виджета, для которого был вызван метод. Это означает, что для нашего главного окна приложения в «системных» областях окна (меню, рамка…) курсор будет иметь форму стрелки.

Qt предоставляет некоторые наиболее используемые курсоры посредством перечисления enumQt::CursorShape. Любую из приведенных в справке констант можно использовать в качестве параметра функции setCursor(). Примеры констант: Qt::CrossCursor, Qt::WaitCursor, Qt::PointingHandCursor,…

**Задание 3**

Установите в конструкторе какой-либо из стандартных курсоров на время работы программы

**на время рисования** изменяйте вид курсора (на паучка).

Подсказка:

в классе Spider заведите переменную типа QCursor, например  
QCursor\*spiCursor;  
а в конструкторе предусмотрите формирование курсора посредством картинки

## Не выпускаем«паука» за пределы окна во время рисования

Попробуйте «потянуть» паутину за пределы окна. Так как Qt автоматически «захватывает» мышь на время нажатия любой ее кнопки, все события мыши поступают виджету, даже если курсор вышел за его пределы. Такой эффект не всегда соответствует логике приложения, поэтому два варианта решения:

В обработчике  
void Spider::mouseMoveEvent(QMouseEvent\*event)  
можно проверить – если мышь вышла за пределы окна с помощью  
bool QRect::contains( const [QPoint](file:///C:\\Users\\user\\AppData\\Roaming\\Microsoft\\Word\\qpoint.html) & *point*, bool *proper* = false ) const

1. способ
   1. вернуть оригинальный курсор
   2. установить признак «Не рисуем»
2. способ

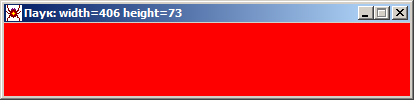
а можно «возвращать» курсор в пределы виджета посредством  
void QCursor::setPos(int x,int y)[static] или  
void QCursor::setPos(const QPoint &p)[static]

Замечание 1: метод ожидает позицию в глобальных координатах. Можно воспользоваться методом mapToGlobal().  
Замечание 2: в ОС Windows есть системная Win32 API функция ClipCursor(), которая «не выпускает» курсор за пределы окна, но в Qt ее аналога нет.

**Задание 4**

Отслеживайте положение курсора относительно виджета и не позволяйте паучку «покидать» пределы рабочей области виджета.

## Отслеживаем изменения размеров окна



Чтобы реагировать на изменение размеров окна – перегружаем виртуальный обработчик:

voidQWidget::**resizeEvent**([QResizeEvent](qthelp://org.qt-project.qtwidgets.532/qtgui/qresizeevent.html) \**event*) [virtual protected]

**Задание 5**

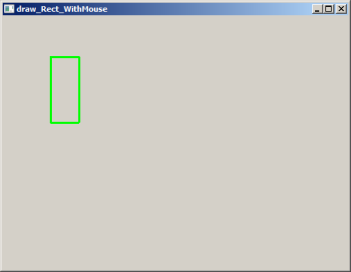
При изменении размера требуется формировать в заголовке окна строку, содержащую информацию о текущей ширине и высоте виджета.

## Рисование прямоугольника с использованием мыши

**Задание 6:**

Нарисуйте на виджете прямоугольник следующим образом:

1. при нажатии правой кнопки мыши определяются координаты одной из вершин.
2. При перемещении мыши (при нажатой кнопке) определяются координаты противоположной вершины и выполняется отрисовка текущего прямоугольника.
3. Рисование прямоугольника завершается, когда кнопка мыши отжимается.



1. Рисование начинается при нажатии кнопки мыши над виджетом.
2. При нажатии кнопки мыши (событие *mousePressEvent*(QMouseEvent \*)):
   1. устанавливается признак «рисуем»: *isDrawing=true*;
   2. фиксируется одна из вершин прямоугольника (запомните координаты в переменной m\_startPoint).
3. при перемещении (событие *mouseMoveEvent*(QMouseEvent \*)) мыши (**только** если происходит рисование):
   1. получаем **текущую** координату курсора – m\_curPoint
   2. выполняем перерисовку виджета
   3. При **отпускании** мыши (**только** если происходит рисование) – «рисование текущего прямоугольника закончено нужно сбросить признак «рисуем»

**Задание дополнительное:**

Попытайтесь доработать программу таким образом, чтобы отображались все нарисованные прямоугольник, а не только самый последний.

Подсказка:

Для того, чтобы отображались ВСЕ нарисованные прямоугольники можно:

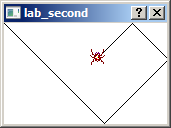
* создать контейнер для хранения прямоугольников
* при завершении формирования прямоугольника сохранять его в контейнере
* в событии paintEvent выполнить рисование всех сохраненных прямоугольников.

## Дополнительно (выполняем в отдельных проектах):

## Таймер

Встроенный таймер:

* В каждом классе, производном от QObject, есть возможность обрабатывать событие таймера. После того, как таймер запущен посредством метода startTimer(), с указанным интервалом будет вызываться перегруженный виртуальный метод timerEvent().
* «Остановить» таймер можно посредством метода killTimer()



**Задание 7.1:**

Реализуйте передвижение паучка по окну.

Начинать и останавливать движение можно по событию нажатия клавиши.

Подсказки:

* При обработке события таймера паучек «передвигается» в определенном направлении на заданные значения перемещения.
* В том случае, когда паучек достигает границы, он должен «отталкиваться» от стенки и менять направление движения.

**Задание 7.2:**

Попробуйте отображать траекторию движения паучка.

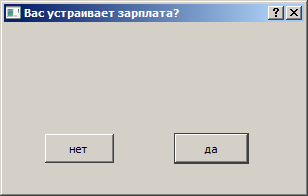
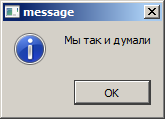
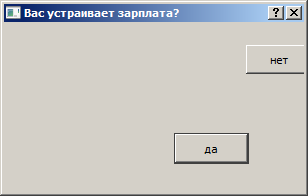
Подсказка:

см. подсказку к заданию 6.

Или

1. В конструкторе виджета создать QPixmap в памяти: *QPixmap(int width, int height);*
2. выполнять рисование каждого отрезка пути на QPixmap;
3. в событии paintEvent выполнить рисование объекта типа QPixmap на виджете.

## События QEvent::Enter, QEvent::HoverEnter, QEvent::HoverMove

* При нажатии на кнопку «Да» выводится информационное сообщение QMessageBox с текстом “Мы так и думали”.
* При наведении на кнопку «Нет» изменяются ее координаты расположения – кнопка «убегает от мышки».

Подсказки (одно из решений):

1. Можно обрабатывать события: QEvent::Enter, QEvent::HoverEnter, QEvent::HoverMove.
2. Чтобы обработать эти события, надо:

* в классе виджета создать фильтр (перегрузить метод *eventFilter()* )
* в котором для кнопки «Нет» обрабатывать перечисленные события («передвигать» кнопку подальше от мышки)
* Для того, чтобы работал фильтр, требуется установить его на кнопку, т.е. метод installEventFilter вызывается для кнопки.