

# Програмування GUI

На основі мови C++ та фреймворку Qt

## Лекція 7

# Події



Обробка подій лежить в основі кожної програми, що має інтерфейс користувача (користувач натиснув на клавішу - буде згенеровано подія клавіатури).



Інформація про створену подію зберігається в об'єкті події.



Базовим класом для всіх класів об'єктів подій є клас `QEvent`.



В Qt є безліч класів для різного роду подій - для промальовування вікон, для миші, клавіатури, таймера і т.д.



# Події

Клас QWidget має безліч віртуальних захищених методів для обробки відповідних подій.

Загальний формат цих методів такий:

protected:

```
virtual void типПодіїEvent (QТипПодіїEvent *  
e) ;
```

е - покажчик на об'єкт події, що містить інформацію про неї.

```
virtual void resizeEvent (QResizeEvent * e) ;  
// подія зміни розмірів вікна
```

# Події



```
#include <QWidget>
#include <QResizeEvent>
#include <QMoveEvent>
#include <QMouseEvent>
#include <QKeyEvent>
#include <QPaintEvent>

class MyWidget : public QWidget
{
    Q_OBJECT

public:
    int timer1, timer2;
    MyWidget(QWidget *parent = nullptr) {setMouseTracking(true);}
    ~MyWidget();

protected:
    void resizeEvent (QResizeEvent *e); // изменение размеров виджета
    void moveEvent (QMoveEvent *e); // перемещение виджета
    void mouseMoveEvent (QMouseEvent *e); // перемещение курсора мыши
    void mousePressEvent (QMouseEvent *e); // нажатие кнопки мыши
    void mouseReleaseEvent (QMouseEvent *e); // отпускание кнопки мыши
    void mouseDoubleClickEvent (QMouseEvent *e); // двойной щелчок мыши
    void keyPressEvent (QKeyEvent *e); // нажатие клавиши клавиатуры
    void keyReleaseEvent (QKeyEvent *e); // отжатие клавиши клавиатуры
    void timerEvent (QTimerEvent *e); // событие таймера
    void paintEvent (QPaintEvent *e); // перерисовка поверхности виджета
```

# QResizeEvent



Подія зміни розмірів екрану



```
void MyWidget::resizeEvent (QResizeEvent *e)
{
    QSize size=e->size(); //новий розмір віджету
    QSize oldSize=e->oldSize(); // старий розмір
    int w=size.width(); // нова ширина
    int h=size.height(); // нова висота
}
```



# QMoveEvent



Подія переміщення віджета

```
void MyWidget::moveEvent (QMoveEvent* e)
```




```
{  
    QPoint pos=e->pos(); // нові координати віджета  
    QPoint oldPos=e->oldPos(); // старі координати  
    int x=pos.x(); // нова горизонтальна координата  
    int y=pos.y(); // нова вертикальна координата  
}
```




# QMouseEvent



 Дана подія виникає при переміщенні курсору миші над поверхнею віджета, натисканні і відпусканні кнопок миші.

 Клас `QMouseEvent` містить наступні функції:

- `Qt::MouseButton button()` – дозволяє

 дізнатися, яка з кнопок миші викликала подію

`Qt::LeftButton`

`Qt::RightButton`

 `Qt::MidButton`

`Qt::NoButton` – значення буде повернуто, якщо подію викликано не кнопкою миші, а, наприклад, при переміщенні курсору.



# MouseEvent



• `Qt::MouseButton buttons()` – дозволяє дізнатися стан кнопок миші на момент виникнення події (наприклад, при переміщенні курсору).



Значення, що повертається, є комбінацією значень (прапорців), об'єднаних операцією OR (`|`). Прапорці будуть включені для натиснутих кнопок і скинуті для відпущених.



• `QPoint pos()` – повертає координати курсора миші щодо верхнього лівого кута клієнтської області віджету.



• `QPoint globalPos()` – повертає екранні координати курсора миші.

• `int x()`, `int y()`, `int globalX()`, `int globalY()` – аналогічно попереднім





# QMouseEvent



`Qt::keyboardModifiers modifiers()` – дозволяє дізнатися стан керуючих клавіш клавіатури на момент генерації події миші. Функція поверне комбінацію прапорців, об'єднаних операцією OR (`|`), які відповідають натисканням клавіш.




`Qt::ShiftModifier` – `<Shift>`

`Qt::ControlModifier` – `<Ctrl>`


`Qt::AltModifier` – `<Alt>`



# QMouseEvent




```
void MyWidget::mouseMoveEvent(QMouseEvent *e)
{ int x=e->globalX(); // екранні координати
  int y=e->globalY(); // курсора миші
```




```
QString str="x= "+QString::number(x,10)+"y=
"+QString::number(y,10);
```

```
bool shiftPress=e->modifiers() &Qt::ShiftModifier;
//shiftPress==true, якщо <Shift> натиснуто
```



```
bool rightPress=e->buttons() &Qt::RightButton;
//rightPress==true, якщо праву кнопку миші натиснуто
```



```
if (shiftPress&rightPress) // якщо <Shift> та праву
                           //кнопку миші натиснуто,
  setWindowTitle(str); //в заголовку віджета будуть
                       //відображатися координати
                       // курсора миші при його переміщенні
```




# QMouseEvent




```
void MyWidget::mousePressEvent (QMouseEvent *e)
```

```
{ if (Qt::LeftButton&e->button())
    setWindowTitle("Left Button Pressed");
else if (Qt::RightButton&e->button())
    setWindowTitle("Right Button Pressed");
}
```



```
void MyWidget::mouseReleaseEvent (QMouseEvent *e)
```



```
{
    if (Qt::LeftButton&e->button())
        setWindowTitle("Left Button released");
}
```



```
void MyWidget::mouseDoubleClickEvent (QMouseEvent *e)
```

```
{
    setWindowTitle("DoubleClick");
}
```



# QKeyEvent



Показчик на об'єкт події клавіатури класу `QKeyEvent` передається в методи обробки подій натискання і віджимання клавіатурних клавіш.

Клас `QKeyEvent` містить наступні функції:

- `int key()` – повертає код клавіші, що натиснуто, в кодуванні Qt. Значення, що повертається, є одним зі значень перерахування `Qt::Key`.

Наприклад, `Qt::Key_A` - код клавіші <A> (регістр не має значення);

`Qt::Key_F1` - <F1>, `Qt::Key_1` - <1>,  
`Qt::Key_Escape` - <Esc>.

# QKeyEvent



- `QString text()` – повертає рядок, що містить символ натиснутої клавіші в кодуванні Unicode. Якщо клавіша не є символьною (наприклад, `<Ctrl>`), поверне порожній рядок.
- `int count()` – повертає число символів натиснутої клавіші, що містяться в об'єкті події, в разі натискання і подальшого утримання символьної клавіші. Значення, яке повертається, дорівнює довжині рядка, що повертається методом `text()`.



# QKeyEvent



```
void MyWidget::keyPressEvent (QKeyEvent *e)
```

```
{
```

```
if (Qt::Key_F1==e->key())
```

```
setWindowTitle("F1");
```

```
else if ("H"==e->text())
```

```
setWindowTitle("Hello");
```

```
else if ("h"==e->text())
```

```
setWindowTitle("hello");
```

```
}
```

```
void MyWidget::keyReleaseEvent (QKeyEvent *e)
```


```
{
```

```
setWindowTitle("Key Released");
```


```
}
```




# QTimerEvent

A small icon of a network or graph structure.


Об'єкт таймера використовується для відліку певного інтервалу часу і генерації події `QTimerEvent` після закінчення інтервалу.

A small icon of a checklist with a green checkmark.

У загальному випадку, подія таймера генерується не один раз, а періодично.

A small icon of a document with a checkmark.

Клас `QTimerEvent` містить метод `int timerId()`, який повертає ідентифікаційний номер таймера.

A small icon of a skull and crossbones.

Кожен клас, успадкований від `QObject`, містить підтримку таймерів.



# QTimerEvent

A small icon of a network or graph structure with nodes and connecting lines.


Запустити таймер можна методом класу `QObject`

```
int startTimer (int msec),
```

де `msec` - задає інтервал таймера в мілісекундах.

A small icon of a clipboard with a checkmark.

Метод повертає ідентифікаційний код таймера, або 0 в разі невдачі створення.

A small icon of a computer monitor.

Ідентифікатор таймера необхідний, якщо в програмі використовується кілька таймерів - для визначення таймера, який згенерував подію.

Знищити таймер можна методом класу `QObject`

A small icon of two interlocking gears.


```
void killTimer (int id), де id -
```

ідентифікатор таймера.






# QTimerEvent



```
void MyWidget::start ()
{
    timer1=startTimer (1000); // інтервал=1 сек
    timer2=startTimer (2000); // інтервал=2 сек
}

void MyWidget::timerEvent (QTimerEvent *e)
{
    if (e->timerId()==timer1) {} //обробка події
    для 1го таймера
    else if (e->timerId()==timer2) { } //обробка
    події для 2го таймера}
}

void MyWidget::stop ()
{
    killTimer(timer1);
    killTimer(timer2);
}
```



# QPaintEvent



Подія перемальовування поверхні віджета.

Дана подія виникає, коли віджет відображається на екрані:

1. Методом `show ()`
2. В результаті методу `update ()` - перемальовування в порядку вилучення події з черги;
3. В результаті методу `repaint ()` - негайне перемальовування;
4. Після перекриття вікна іншим вікном;
5. І т.д.

# QPaintEvent

Клас QPaintEvent містить методи:

- `QRect rect()`
- `QRegion region()`.

Вони дозволяють визначити координати і розміри області, що вимагає перемальовування.

Малювати на поверхні віджета можна за допомогою об'єкта класу QPainter (т.зв. малювальник).

Цей клас містить функції малювання точки, кривої, прямий, різних фігур.

# QPaintEvent



Малювання за допомогою даного об'єкта може виконуватися на об'єктах, успадкованих від класу `QPaintDevice: QWidget, QPicture, QPixmap`.

Для використання об'єкта класу `QPainter`, необхідно передати адресу об'єкта, на якому буде проводитися малювання. Це можна зробити

- За допомогою конструктора класу `QPainter`
- Викликом методу `bool begin (QPaintDevice * device)` класу `QPainter`.



# QPaintEvent



Конструктори класу `QPainter`:

- `QPainter ()`
- `QPainter (QPaintDevice *d)`

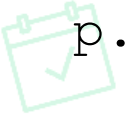
Після закінчення малювання необхідно викликати метод `bool end ()` класу `QPainter`. Це необхідно, оскільки всі команди малювання надходять в чергу подій віджета і обробляються в порядку вилучення з черги.

При виклику методу `end ()`, звільняються всі ресурси, необхідні і раніше виділені для малювання (тобто більше малювати не можна). Також відбувається автоматичне виконання всіх невиконаних команд, що залишилися в черзі.

# QPaintEvent



```
void MyWidget::paintEvent (QPaintEvent *e)
{
    QPainter p;
    p.begin(this); //передає адреса об'єкта, на
                  // поверхні якого буде малювати
    p.drawText (50,50, "Hello");// команда малювання
                              // виведення тексту
    p.end(); // закінчення малювання
```



# Перехоплення подій



В Qt існує можливість одним об'єктом класу QObject або його спадкоємців (наприклад, QWidget)



перехоплювати події, адресовані іншому об'єкту класу QObject або його спадкоємцям.





Перехоплення подій можна реалізувати за допомогою **фільтра подій.**



# Перехоплення подій

 Перехоплення подій реалізується в 2 етапи:

-  1. Необхідно викликати функцію `void installEventFilter (QObject * obj)` цільовим об'єктом - тобто тим об'єктом, у якого перехоплюватимуться події. `obj` - покажчик на об'єкт-перехоплювач.
-  2. Перевизначити потрібним чином віртуальний захищений метод `bool eventFilter (QObject * obj, QEvent * e)` об'єкта перехоплювача. `obj` - покажчик на цільовий об'єкт, `e` - покажчик на об'єкт подій, що містить інформацію про подію. `QEvent` - клас події, що є базовим для всіх класів подій Qt.





# Перехоплення подій



Клас QEvent надає метод `type ()`, який повертає тип події - миші, клавіатури, таймер і т.д.



Перший етап перехоплення події як правило реалізується в конструкторі об'єкта-перехоплювача.



# Перехоплення подій



Приклад.



На головному вікні розташовані 3 однорядкових редактора. Необхідно реалізувати можливість передачі фокусу введення між редакторами за допомогою клавіш курсору  $\leftarrow \rightarrow$   $\leftarrow \rightarrow$   $\leftarrow \uparrow$   $\leftarrow \downarrow$ . Головне вікно буде об'єктом-перехоплювачем, текстові редактори - цільовими об'єктами.



# Перехоплення подій



```
MyWidget::MyWidget(QWidget *parent = nullptr) {...
edit1->installEventFilter(this);
edit2->installEventFilter(this);
edit3->installEventFilter(this);}

bool MyWidget::eventFilter(QObject *obj, QEvent *e)
{ if (e->type()==QEvent::KeyPress) //если тип - событие клавиатуры
{//обрабатываем
if (obj==edit1||obj==edit2||obj==edit3) //проверяем - целевой объект
//один из трех редакторов?
{//обрабатываем
QKeyEvent *keyEvent=static_cast<QKeyEvent*>(e);
//static_cast<> - оператор преобразования типов
//тип должен быть указателем или ссылкой на объект.
//e - преобразуемый указатель, keyEvent - результат преобразования
//Приведение необходимо, т.к. класс QEvent не содержит метода key(),
int key=keyEvent->key();
if (Qt::Key_Left==key||Qt::Key_Up==key)
{focusPreviousChild(); //передать фокус ввода предыдущему дочернему окну
return true; }
if (Qt::Key_Right==key||Qt::Key_Down==key)
{ focusNextChild(); //передать фокус ввода следующему дочернему окну
return true; } } }
return QWidget::eventFilter(obj,e); //передача события на обработку
//стандартному обработчику перехвата событий класса QWidget }
```