Semnale si sloturi

- Semnalele si sloturile sunt folosite in Qt pentru comunicare intre obiecte
- Cand facem modificari la o componenta (scriem un text, apasam un buton, selectam un element, etc.) dorim ca alte parti ale aplicatiei sa fie notificate (sa actualizam alte componente, sa executam o metoda, etc). De exemplu : Daca utilizatorul apasa pe butonul **Close**, dorim sa inchidem fereastra, adica sa apelam metoda **close()** a ferestrei.
- In general bibliotecele pentru interfete grafice folosesc callback pentru aceste interactiuni.
- Callback
  - este un pointer la o functie
  - daca intr-o metoda dorim sa notificam aparitia unui eveniment, putem folosi un pointer la o functie (callback, primit ca parametru)
- Dezavantaje callback in C++ :
  - daca avem mai multe evenimente de notificat, ne trebuie functii separate callback sau sa folosim parametrii generici (void*) care nu se pot verifica la compilare
  - metoda care apeleaza callback este cuplata tare de callback (trebuie sa stie signatura functiei, parametrii. Are nevoie de referinta la metoda callback)

Signal. Slot.

- Semnalul este emis (ex. : &QPushButton::clicked) la aparitia unui eveniment
- Componentele Qt (widget) emit semnale pentru a indica schimbari de stari generate de utilizator
- Un **slot** este o functie care este apelata ca si raspuns la un semnal.
- Semnalul se poate conecta la un slot, astfel la emiterea semnalului slotul este automat executat.

```
QApplication a(argc, argv);
QPushButton* btn = new QPushButton("&Close");
QObject::connect(btn, &QPushButton::clicked, &a, &QApplication::quit);
btn->show();
```

- **Slot** poate fi folosit pentru a reactiona la semnale, dar ele sunt de fapt metode normale sau functii lambda.
- Semnalele si sloturile sunt decuplate intre ele
  - Obiectele care emit semnale nu au cunostinte despre sloturile care sunt conectate la semnal
  - Slotul nu are cunostinta despre semnalele conectate la el
  - Decuplarea permite crearea de componente cu adevarat independente folosind Qt
- In general componentele Qt au un set predefinit de semnale. Se pot adauga si semnale noi folosind mostenire (clasa derivata poate adauga semnale noi)

Conectarea semnalelor cu sloturi

Folosind metoda QObject::connect putem conecta semnale si sloturi

```
QWidget* createButtons(QApplication &a) {
    QWidget* btns = new QWidget();
    QHBoxLayout* btnsLayout = new QHBoxLayout();
    QPushButton* store = new QPushButton(text: "&Store");
    btnsLayout->addWidget(store);
    QPushButton* close = new QPushButton(text: "&Quit");
    btnsLayout->addWidget(close);
    QObject::connect(close, signal: &QPushButton::clicked, &a, slot: &QApplication::quit);

    btns->setLayout(btnsLayout);
    return btns;
}
```

In urma conectarii - slotul este apelat in momentul in care se genereaza semnalul. Slotuile sunt functii normale, apelul este la fel ca la orice functie C++. Diferenta intre un slot si o functie este ca slotul se poate conecta la semnale.

La un semnal se pot conecta mai multe sloturi, in urma emiterii semnalului se vor apela sloturile in ordinea in care au fost conectate.

Conectarea semnalelor cu sloturi - referinta metode, lambda
Inainte de Qt 5 semnalele si sloturile se indicau folosind macrourile SIGNAL, SLOT
Odata cu Qt 5 este permis folosirea de functii lambda, referinte la functii/metode

```
QObject::connect(close, &QPushButton::clicked,
                 &a,QApplication::quit);
```

```
QObject::connect(close, &QPushButton::clicked, [&a]() {
          a.quit();
     });
```

```
QObject::connect(close, SIGNAL(clicked()), &a, SLOT(quit()));
```

&PushButton::clicked este o referinta la functia clicked, metoda din clasa QPushButton.

Folosind functii lambda si referinte la functii compilatorul poate verifica daca semnalul si slotul este compatibil.

Conectarea semnalelor cu sloturi

```
QSpinBox *spAge = new QspinBox();
QSlider *slAge = new QSlider(Qt::Horizontal);
```

```
//Synchronise the spinner and the slider
//Connect spin box - valueChanged to slider setValue
QObject::connect(spAge, &QSpinBox::valueChanged, slAge,
&QSlider::setValue);
//Connect - slider valueChanged to spin box setValue
QObject::connect(slAge,
SIGNAL(valueChanged(int)),spAge,SLOT(setValue(int)));
```

```
//prutem prelua valori de la semnal
QObject::connect(spAge, &QSpinBox::valueChanged,
                                    slAge, &QSlider::setValue);
//Connect - slider valueChanged to spin box setValue
QObject::connect(slAge, &QSlider::valueChanged, [spAge](int val) {
          spAge->setValue(val);}
);
```

Daca utilizatorul schimba valoarea in spAge (Spin Box) :

- Se emite semnalul valueChanged(int) argumentul primeste valoarea curenta din spinner.
- Fiindca cele doua componente (spinner, slider) sunt conectate se apeleaza metoda setValue(int) de la slider.
- Argumentul de la metoda valueChanged se transmite ca si parametru pentru slotul, metoda setValue din slider.
- Sliderul se actualizeaza pentru a reflecta valoarea primita prin setValue si emite la randul lui un semnal valueChanged

Componente definite de utilizator

- Se creeaza clase separate pentru interfata grafica utilizator
- Componentele grafice create de utilizator extind componentele existente in Qt
- Scopul este de a crea componente independente cu semnale si sloturi proprii

```
/**
 * GUI for storing Persons
 */
class StorePersonGUI: public QWidget {
public:
     StorePersonGUI();
private:
     QLabel *lblName;
     QLineEdit *txtName;
     QLabel *lblAdr;
     QLineEdit *txtAdr;
     QSpinBox *spAge;
```

```
    QLabel *lblAge2;
    QSlider *slAge;
    QLabel *lblAge3;
    QPushButton* store;
    QPushButton* close;
    /**
     * Assemble the GUI
     */
    void buildUI();
    /**
     * Link signals and slots to define the behaviour of the GUI
     */
    void connectSignalsSlots();
};
```

QMainWindow

- In functie de componenta pe care o definim putem extinde clasa QWidget, QMainWindow, QDialog.
- Clasa QMainWindow poate fi folosita pentru a crea fereastra principala pentru o aplicatie cu interfata grafica utilizator.
- QMainWindow are propriul layout, se poate adauga toolbar, meniuri, status bar.
- QMainWindow defineste elementele de baza ce apar in mod general la o aplicatie.

Layout QMainWindow:

- Meniu - pe partea de sus

```
QAction *openAction = new QAction("&Load", this);
QAction *saveAction = new QAction("&Save", this);
QAction *exitAction = new QAction("E&xit", this);
fileMenu = menuBar()->addMenu("&File");
fileMenu->addAction(openAction);
fileMenu->addAction(saveAction);
fileMenu->addSeparator();
fileMenu->addAction(exitAction);
```

- Toolbar

```
QToolBar* fileToolBar = addToolBar("&File");
fileToolBar->addAction(openAction);
fileToolBar->addAction(saveAction);
```

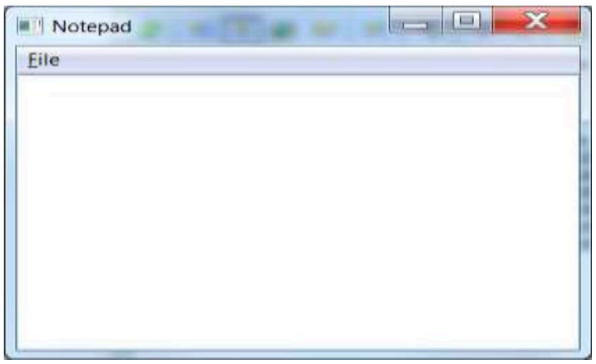
- Componenta din centru

```
    middle = new QTableWidget(10, 10, this);
    setCentralWidget(middle);
```

- Status bar - in partea de jos

```
statusBar()->showMessage("Status Message ....");
```

Notepad



```
class Notepad : public QMainWindow
{

public:
    Notepad();

private:
    void open();
    void save();
    void quit2();
```

```

    openAction = new QAction(tr("&Load"), this);
    saveAction = new QAction(tr("&Save"), this);
    exitAction = new QAction(tr("E&xit"), this);

    connect(openAction, &QAction::triggered, this, &Notepad::open);
    connect(saveAction, &QAction::triggered, this, &Notepad::save);
    connect(exitAction, &QAction::triggered, this, &Notepad::quit2);
```

```
void Notepad::open()
{
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open File"), "",
        tr("Text Files (*.txt);;C++ Files (*.cpp *.h)"));

    if (fileName != "") {
        QFile file(fileName);
        if (!file.open(QIODevice::ReadOnly)) {
            QMessageBox::critical(this, tr("Error"), tr("Could not open file"));
            return;
        }
        QTextStream in(&file);
        textEdit->setText(in.readAll());
        file.close();
    }
```

Exemplu PetStore

```
class PetStoreGUI : public QWidget{
private:
	PetController& ctr;
	QListWidget* lst;
	QPushButton* btnSortByPrice;
	QPushButton* btnSortByType;
	QLineEdit* txtSpecies;
	QLineEdit* txtType;
	QLineEdit* txtPrice;
	void initGUICmps();
	void connectSignalsSlots();
	void reloadList(std::vector<Pet> pets);
public:
	PetStoreGUI(PetController& ctr) :ctr{ ctr } {
		initGUICmps();
		connectSignalsSlots();
		reloadList(ctr.getAllPets());
	}
};
void PetStoreGUI::reloadList(std::vector<Pet> pets) {
	lst->clear();
	for (auto& p : pets) {
		QListWidgetItem* item = new QListWidgetItem(p.getSpecies(), lst);
		item->setData(Qt::UserRole, p.getType());//adaug in lista (invizibil) si type
		//lst->addItem(p.getSpecies());
	}
}
void PetStoreGUI::connectSignalsSlots() {
	//cand se emite semnalul clicked de pe buton reincarc lista
	QObject::connect(btnSortByPrice, &QPushButton::clicked, [&]() {
		reloadList(ctr.getSortByPrice());
	});
	//cand se emite semnalul clicked de pe buton reincarc lista
	QObject::connect(btnSortByType, &QPushButton::clicked, [&]() {
		reloadList(ctr.getSortByType());
	});
	//cand se selecteaza elementul din lista incarc detaliile
	QObject::connect(lst, &QListWidget::itemSelectionChanged, [&]() {
		if (lst->selectedItems().isEmpty()) {
			//nu este nimic selectat (golesc detaliile)
			txtSpecies->setText("");
			txtType->setText("");
			txtPrice->setText("");
			return;
		}
		QListWidgetItem* selItem =  lst->selectedItems().at(0);
		txtSpecies->setText(selItem->text());
		txtType->setText(selItem->data(Qt::UserRole).toString());
	});
}
```

Clase QT - utile

Qstring - sir de caractere Unicode

Metodele care primesc un parametru QString accepta si char* (QString are un constructor cu char*)

```
txtSpecies->setText("ceva text");
```

Exista metode de a transforma din QString in std::string si invers

```
QString s = "145";
std::string ss = s.toStdString();
QString s2 = QString::fromStdString(ss);
```

Exista si posibilitatea de a transforma numere in QString si invers

```
QString s = "145";
int a = s.toInt();
double d = 3.8;
QString s3 = QString::number(d);
```

QList/QVector lista/vector variante Qt pentru containere din STL

- au operatii similare ca cele din stl : at(0), pop(), isEmpty(), size(), etc.

Se pot transforma in variantele stl :

```
QList<int> l;
l.push_back(3);
l.push_front(4);
l[1] = 4; l.at(0);
```

```
std::list<int> ll = l.toStdList();
QList<int> l2 = QList<int>::fromStdList(ll);
```

QMessageBox

Putem afisa o fereastra cu informatii utile

```
QMessageBox::information(this, "Info", "Selection changed");//afiseaza mesaj

int ret = QMessageBox::warning(this, "My Application",
             "The document has been modified.\nDo you want to save your changes?",
             QMessageBox::Save | QMessageBox::Discard| QMessageBox::Cancel,
             QMessageBox::Save);
if (ret == QMessageBox::Save) {
    //do save
}
```

QDebug

Stream pentru a scrie informatii ce ajuta la depanarea programelor

```
qDebug() << "Date:" << QDate::currentDate();
```

Item based Widgets - QListWidget, QTableWidget

QListWidget / QListWidgetItem	QTableWidget / QTableWidgetItem

`QListWidget* lst = new QListWidget;`
 	
`//se pot adauga elemente`
`QListWidgetItem* item = new`
`QListWidgetItem("Bla", lst);`
	`//se creaza`
	`int nrLinii = 4;`
	`int nrColoane = 3;`
	`QTableWidget* tbl = new`
	`QTablWidget{ nrLinii,nrColoane };`
	
	`//se pot adauga elemente`
	`QTableWidgetItem* cellItem1 = new`
	`QTableWidgetItem("Linie1");`
	`tbl->setItem(0, 0, cellItem1);`
	`tbl->setItem(0, 1, new`
	`QTableWidgetItem("Linie1`
	`coloana2"));`

`//se poate configura modul de selectie`
`lst->setSelectionMode(`
`QAbstractItemView::SingleSelection);`
`//se poate obtine selectia`
`auto selItms = lst->selectedItems();`
	`//se poate configura modul de selectie`
	`tbl->setSelectionBehavior(`
	`QAbstractItemView::SelectRows);`
	`tbl->setSelectionMode(`
	`QAbstractItemView::SingleSelection);`
	`//se poate obtine selectia`
	`auto selTblItms = tbl->selectedItems();`

`//putem reactiona la semnale`
`QObject::connect(lst,`
`&QListWidget::itemSelectionChanged,`
`[lst]() {`
`qDebug() << "Selectie \n" <<`
`lst->selectedItems() << "\n";`
`});`
	`//putem reactiona la semnale`
	`QObject::connect(tbl,`
	`&QTableWidget::itemSelectionChanged,`
	`[tbl]() {`
	`qDebug() << "Selectie tabel\n"<<`
	`tbl->selectedItems()<<"\n";`
	`});`

Fiecare celula din tabel / lista contine textul dar si alte informatii :

```
      //informatii suplimentare in item
       item->setBackground(QBrush{ Qt::red, Qt::SolidPattern });
       item->setForeground(Qt::blue);
       item->setData(Qt::UserRole, QString{ "informatii care nu se vad" });
       item->setCheckState(Qt::Checked);
       item->setIcon(QApplication::style()->standardIcon(QStyle::SP_BrowserReload));
```

```
//informatii suplimentare pentru fiecare celula
       cellItem1->setBackground(QBrush{ Qt::red, Qt::SolidPattern });
       cellItem1->setForeground(Qt::blue);
       cellItem1->setData(Qt::UserRole, QString{ "informatii care nu se vad" });
       cellItem1->setCheckState(Qt::Unchecked);
       cellItem1->setIcon(QApplication::style()->standardIcon(QStyle::SP_ArrowBack));
```

Qt Build system

O aplicatie C++ contine fisiere header (.h) si fisiere (.cpp)
Procesul de build pentru o aplicatie C++ :

- Se compileaza fisierele cpp folosind un compilator (fisierele sursa pot referi alte fisiere header) → fisiere obiect (.o)
- Folosind un linker, se combina fisierele obiect (link edit) → fisier executabil (.exe)

Qt introduce pasi aditionali :

Meta-object compiler

Compilatorul ia toate clasele care incep cu macroul Q_OBJECT si genereaza fisiere sursa (.cpp). Aceste fisiere sursa contin informatii despre clasele compilate (nume, ierarhia de clase) si informatii despre signale si sloturi. Practic in fisierele sursa generate gasim codul efectiv care apeleaza metodele slot cand un semnal este emis.

User interface compiler

Compilatorul pentru interfete grafice are ca intrare fisiere create de Qt Designer si genereaza cod C++

Semnale proprii

Folosind macroul signals se pot declara semnale proprii pentru componentele pe care le cream

```
signals:
     void storeRq(QString* name,QString* adr );
```

Cuvantul rezervat **emit** este folosit pentru a emite un semnal

```
emit storeRq(txtName->text(),txtAdr->text());
```

Semnale proprii exemplu

```
class BrickGameEngine: public QObject{
      Q_OBJECT  //e nevoie de acest macro daca vrem semnale custom
      int score = 0;
      int dead = 0;
      int nrBricks = 0;
      QTimer timer;
      int ballMoveDelay = 20;
      int elapsedMoves = 0;
signals:
      //semnale generate de engine
      void scoreChanged(int currentScore);
      void deadChanged(int currentNrDead);
      void advanceBoard();
      void brickCreated(int x, int y, int brickW, int brickH);
      void gameFinished(bool win);

public:
      BrickGameEngine() {
            emit scoreChanged(score);
            emit deadChanged(dead);
      }

      void brickHit() {
            score += 1;
            nrBricks--;
            emit scoreChanged(score);
      }
QObject::connect(&engine, &BrickGameEngine::gameFinished, [&](bool win) {
                  if (win) {
                        QMessageBox::information(this, "Info","You win!!!");
                  }
                  else {
                        QMessageBox::information(this, "Info", "You lose!!!");
                  }
            });
```

Desenare low-level
Clasa **QPainter** permite desenarea "manuala", are functii optimizate pentru a desena orice elemente avem nevoie intr-o aplicatie grafica (linii, dreptunghiuri, elipse, imagini, etc)

In general obiectul QPainter se foloseste in interiorul metodei pentEvent, metoda ce este apelata de fiecare data cand s-a cerut redesenarea widgetului.

```
void paintEvent(QPaintEvent* ev) override {
            QPainter p{ this };

            p.drawLine(0, 0, width(), height());
            p.drawImage(x,0,QImage("sky.jpg"));

            p.setPen(Qt::blue);
            p.setFont(QFont("Arial", 30));
            p.drawText(rect(), Qt::AlignTop | Qt::AlignHCenter, "Qt QPainter");

            p.fillRect(0, 100, 100, 100,Qt::BrushStyle::Dense1Pattern);

      }
```

Graphics View Framework
Graphics View ofera suport pentru aplicatii care gestioneaza si interactioneaza cu un numar mare de obiecte 2D.

QGraphicScene - contine elementele grafice **QGraphicItem**, se ocupa cu propagarea evenimentelor catre obiectele grafice 2D.

QGraphicItem - si clasele derivate din el (**QGraphicsRectItem QGraphicsElipseItem QGraphicsTextItem QGraphicsRectItem**) sunt elementele grafice ce se pot adauga intr-o scena.

```
      QGraphicsView* view = new QGraphicsView;
      QGraphicsScene* scene = new QGraphicsScene;
      view->setScene(scene);
      view->setFixedSize(800, 600);
      scene->setSceneRect(0, 0, 800, 600);

      //add items to scene
      QGraphicsEllipseItem* ball = new QGraphicsEllipseItem{ 0,0,20,20 };
      ball->setPos(scene->width() / 2, scene->height()/2 );
      scene->addItem(ball);

      QGraphicsRectItem* r = new QGraphicsRectItem{ 0,0,40,30 };
      r->setPos(20, scene->height() / 2);
      r->setBrush(QBrush(QImage("wood1.jpg")));
      scene->addItem(r);

      view->show();
```

```
class BrickGame:public QGraphicsView {
       QGraphicsScene*  scene;
       Paddle* player;
       Ball* ball;
public:
       BrickGame() {
              setMouseTracking(true);
              initScene();
              createPlayer();
              loadLevel();
              addBall();
              startGame();
       }
void startGame() {
    QTimer* timer = new Qtimer;
    //advanceGame invoked every time
    QObject::connect(timer, &Qtimer::timeout,
                this,&BrickGame::advanceGame);
     //generate timeot signal every 20ms
     timer->start(20);
   }

//handle mouse move
void mouseMoveEvent(QMouseEvent* ev) override{
      //works only if setMouseTracking(true);
      auto x = ev->pos().x();
      player->setPos(x, player->y());

}
}
```

```
class Ball :public QGraphicsEllipseItem {
private:
       QGraphicsDropShadowEffect * effect;
       int dx = 5;
       int dy = -5;
public:
   Ball() {
     setRect(0, 0, 20, 20);
     setBrush(QBrush(QImage("ball.jpg")));

     effect = new QGraphicsDropShadowEffect();
     effect->setBlurRadius(30);
     setGraphicsEffect(effect);
}

  void move() {
     setPos(x() + dx, y() + dy);
    setTransformOriginPoint(rect().width() / 2,
                      rect().height() / 2);
     setRotation(rotation() + 45);
    }

    void changeXDir() {
              dx *= -1;
    }
}
```