Практическое задание №1.

Модификация приложения калькулятора в среде QtCreator.

Цель работы: познакомиться со средой разработки QtCreator, изучить механизм сигналов и слотов, а также работу с QAction.

Задание:

Клонировать open-source проект https://github.com/vpapadopou/qt-simple-calculator и добавить новый функционал:

- Пункт главного меню «Сохранить значение», который сохраняет значение из дисплея (displayPanel) в список истории.
- Отображение временного сообщения в StatusBar о сохранении значения.
- Пункт меню «История» с подпунктами «Показать», «Скрыть», который открывает и закрывает боковое окно с сохранёнными значениями и кнопками управления «скрыть» и «очистить» историю.
- Пункт меню «Справка» для вывода информации о работе калькулятора и сведения об авторах.

Порядок выполнения:

1) Убедитесь, что проект калькулятора собран и работает.
2) Добавьте строку состояния (statusBar) и панель меню (menuBar) на форму (кликнуть правой кнопкой мыши в режиме «Дизайнера» на форме и выбрать «добавить строку состояния», «добавить панель меню»). Создайте пункты меню по заданию:

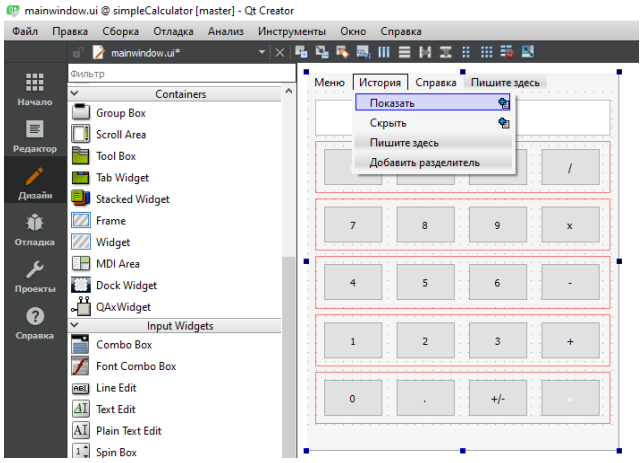

Рисунок 1. Добавление пунктов меню в режиме дизайнера.

Затем задайте новые названия объектам меню в редакторе свойств:

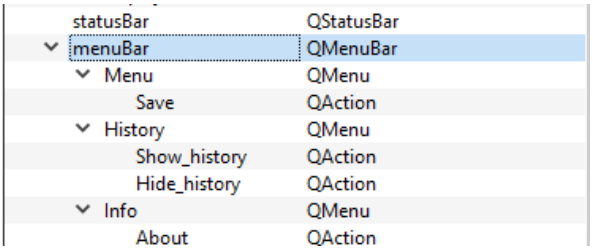

Рисунок 2. Структура меню в инспекторе объектов.

3) Добавьте определение слотов для обработки пунктов меню в файл mainwindow.h:

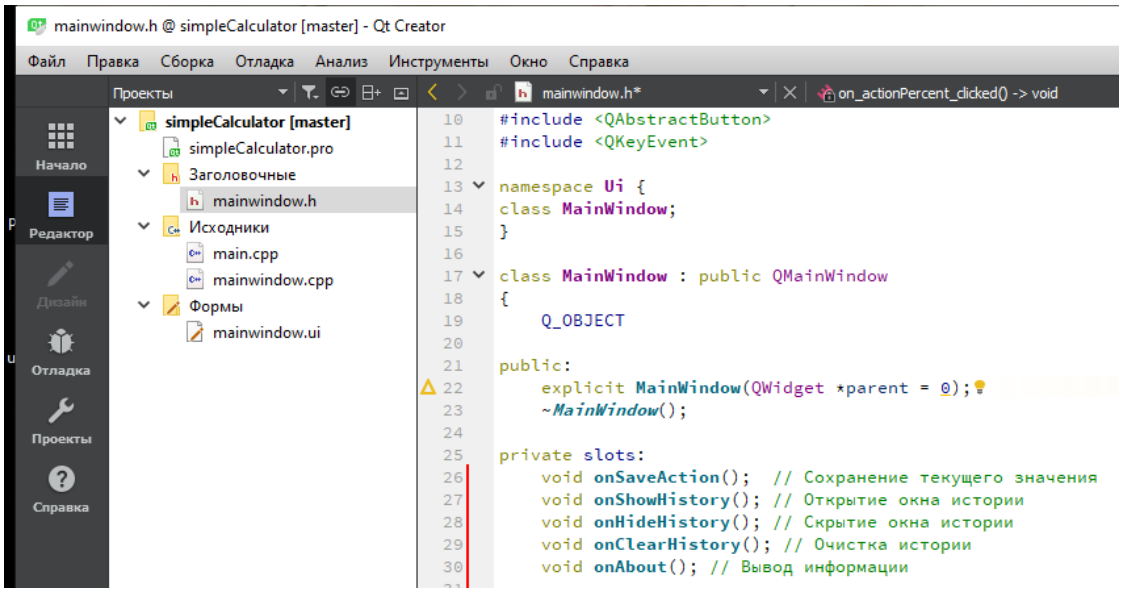

Рисунок 3. Определение слотов.

Также необходимо добавить объявления списка строк (QList для хранения истории) и указателей на элементы интерфейса виджета истории, который далее будет создан. Для временного показа сообщений в statusBar также понадобится объявить указатель на таймер:

```
// Хранение истории вычислений
QList<QString> historyList;

// Виджеты для окна истории
QWidget *historyWidget;
QListWidget *historyView;
QPushButton *hideHistoryButton;
QPushButton *clearHistoryButton;
QVBoxLayout *historyLayout;

// Таймер для исчезновения сообщения
QTimer *statusBarTimer;
```

Рисунок 4. Объявление переменных для работы с историей вычислений

4) В конструкторе MainWindow добавьте создание виджета истории с необходимыми элементами управления. Виджет должен удовлетворять следующим требованиям:

- динамическое создание виджета в коде;
- виджет должен отображаться рядом с основным интерфейсом калькулятора;
- при показе виджета главное окно приложения должно расширяться, а при скрытии – возвращаться к исходной ширине;
- виджет должен содержать список для отображения истории вычислений;
- виджет должен иметь кнопки «скрыть» и «очистить»;
- использование компоновщика;
- нажатие на запись в истории должно обновлять значение на дисплее калькулятора;
- при внесении значения в историю должно отображаться временное сообщение в statusBar;
- пустое значение в список истории не заносится, о чем необходимо уведомлять пользователя в statusBar.

Пример создания виджета истории с использованием вертикального компоновщика:

```
// Создаем виджет истории
historyWidget = new QWidget(this);
historyWidget->setWindowTitle("История");
historyWidget->resize(200, 400);

// Создаем элементы управления
historyView = new QListWidget(historyWidget);
hideHistoryButton = new QPushButton("Скрыть", historyWidget);
clearHistoryButton = new QPushButton("Очистить", historyWidget);

// Вертикальный layout
QVBoxLayout *mainLayout = new QVBoxLayout(historyWidget);
mainLayout->addWidget(hideHistoryButton);
mainLayout->addWidget(clearHistoryButton);
mainLayout->addWidget(historyView);
historyWidget->setLayout(mainLayout);
```

Рисунок 5. Создание виджета истории.

То есть список значений истории (historyList) теперь будет выведен в виджет списка, что допустимо для простых случаев. Для большей гибкости можно воспользоваться паттерном Model/View, о котором речь пойдет позже.

5) В файле mainwindow.cpp добавьте слоты для обработки нажатия кнопок «Скрыть» и «Очистить» для виджета истории:

```
void MainWindow::onHideHistory() {
    if (!historyWidget->isVisible()) return;
    // Возвращаем окно к исходной ширине
    resize(width() - historyWidget->width(), height());
    // Скрываем виджет истории
    historyWidget->hide();
    }

void MainWindow::onClearHistory() {
    historyList.clear();
    historyView->clear();
    ui->statusBar->showMessage("История очищена", 3000);
}
```

Рисунок 6. Слоты для кнопок «Скрыть» и «Очистить».

6) Добавьте слот для обработки нажатия на значение из списка истории:

```
void MainWindow::onHistoryItemClicked(QListWidgetItem *item) {
    QString value = item->text();
    // Устанавливаем новое значение
    ui->displayPanel->setText(value);
}
```

Рисунок 7. Слот обработки нажатия на список значений истории.

7) На данном этапе у вас создан виджет истории и подготовлены слоты для обработки действий пользователя. Далее необходимо настроить сигнально-слотовые связи для виджета истории. Для этого воспользуйтесь функцией connect(): первым параметром укажите элемент интерфейса, который генерирует сигнал, вторым параметром – событие, которое отправляет указанный элемент (сигнал), третьим – объект-получатель сигнала (this ссылается на текущий экземпляр класса), а четвертым – слот для обработки события:

```
// Подключаем сигналы от элементов истории к соответствующим слотам
connect(hideHistoryButton, &QPushButton::clicked, this, &MainWindow::onHideHistory);
connect(clearHistoryButton, &QPushButton::clicked, this, &MainWindow::onClearHistory);
connect(historyView, &QListWidget::itemClicked, this, &MainWindow::onHistoryItemClicked);
```

Рисунок 8. Сигнально-слотовые связи.

8) Чтобы виджет истории корректно отображался на форме, можно воспользоваться контейнерным виджетом QSplitter. Этот виджет позволяет размещать в себе несколько виджетов рядом друг с другом и не только меняет размеры динамически в зависимости от содержимого, но и позволяет регулировать и изменять размер компонентов вручную. Использование компоновщика QHBoxLayout также допускается, но в данном случае будет менее удобным из-за автоматического выравнивания.

   Чтобы иметь возможность менять размеры главного окна, необходимо в режиме дизайнера кликнуть на форму и, убедившись, что в инспекторе объектов выбрано MainWindow, поменять значение свойства sizePolicy на Preferred или Expanding. Затем в конструкторе MainWindow добавить код для создания сплиттера и разместить в нем центральный виджет (centralWidget), содержащий первоначальный калькулятор и созданный виджет истории. А с помощью функции setStretchFactor() можно указать, как дочерние виджеты внутри сплиттера будут растягиваться при изменении окна: нам нужно, чтобы виджет с калькулятором не менял своего размера. После этого в качестве центрального виджета установите созданный сплиттер:

```
QSplitter *mainSplitter = new QSplitter();
mainSplitter->addWidget(ui->centralWidget);
ui->centralWidget->setMinimumWidth(300);
mainSplitter->addWidget(historyWidget);
mainSplitter->setStretchFactor(0,1);
setCentralWidget(mainSplitter);
```

Рисунок 9. Создание сплиттера.

9) Последним действием конструктора скройте виджет истории с помощью функции hide().

10) Вернемся к главному меню программы. Вам необходимо реализовать оставшиеся действия для обработки нажатия на пункты меню: сохранение текущего значения истории, открытие окна истории и вывод информации о программе.

Для корректного сохранения значения истории, необходимо взять его из виджета displayPanel (QLabel) и внести в список истории, если значение не пустое. Также следует выводить временные сообщения в строке статуса. Например, реализация этих действий может выглядеть следующим образом:

```
void MainWindow::onSaveAction() {
    QString currentValue = ui->displayPanel->text();

    if(currentValue.isEmpty()) {
        ui->statusBar->showMessage("Значение пустое не добавлено в историю", 3000);
        return;
    }

    historyList.append(currentValue);
    ui->statusBar->showMessage("Значение " + currentValue + " добавлено в историю", 3000);

    // таймер для автоматического скрытия сообщения
    statusBarTimer = new QTimer(this);
    statusBarTimer->setSingleShot(true);
    connect(statusBarTimer, &QTimer::timeout, this, &MainWindow::clearStatusBarMessage);
    statusBarTimer->start(3000);
    updateHistoryList();
}

void MainWindow::clearStatusBarMessage() {
    ui->statusBar->clearMessage();
}

void MainWindow::updateHistoryList() {
    historyView->clear();
    for (const QString& item : qAsConst(historyList)) {
        historyView->addItem(item);
    }
}
```

Рисунок 10. Действия для пункта меню «Сохранить значение».

Теперь калькулятор в развернутом виде должен выглядеть так:

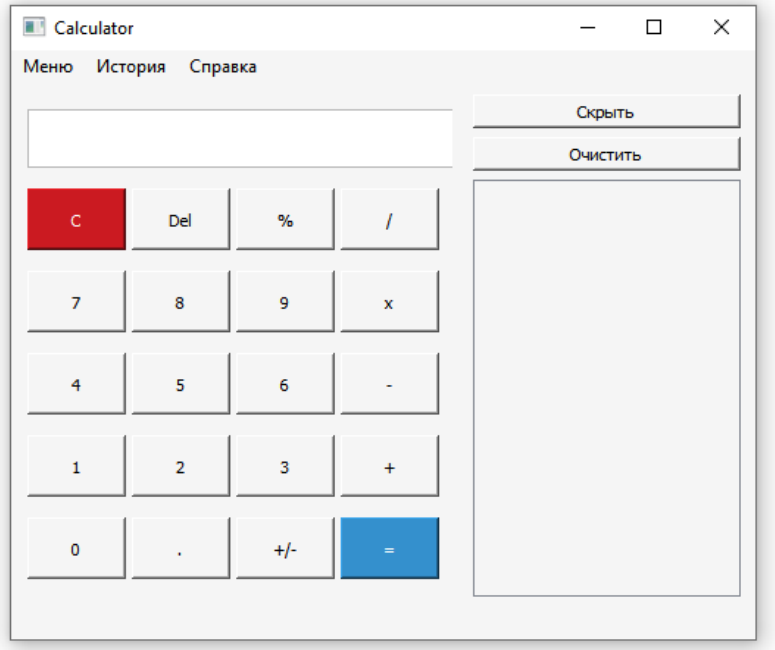

Рисунок 11. Калькулятор с историей

11) Перейдем к обработке нажатия пункта меню «О программе». Недостаточно просто вывести сообщение о программе – необходимо добавить новую форму для показа справки в удобном для пользователя виде. Окно справки должно содержать в себе следующую информацию:

- инструкцию по использованию калькулятора;
- инструкцию по работе с историей;
- описание горячих клавиш;
- указание лицензии и авторства.

Итак, создайте новый класс формы в Qt Designer. Для этого в списке файлов проекта находим «Формы», правой кнопкой мыши выбираем «Добавить новый», затем выбираем «Класс формы Qt Designer».

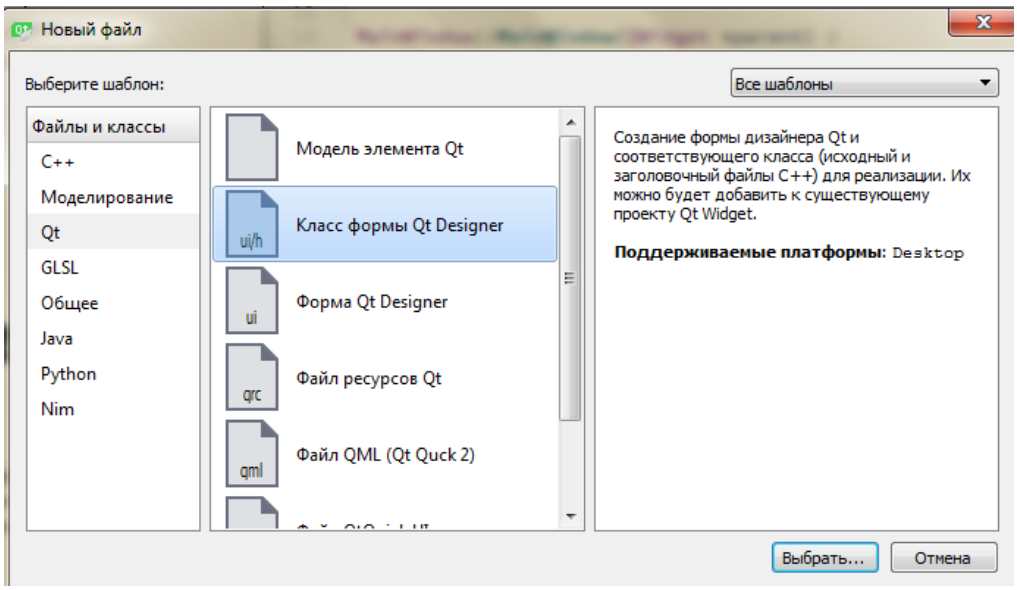

Рисунок 12. Создание класса формы

Например, пусть класс называется Form, реализуем обработчик нажатия пункта меню «О программе»:

```
void MainWindow::onAbout() {
    Form *info = new Form();
    info->show();
}
```

Рисунок 13. Обработчик нажатия пункта меню «О программе»

Таким образом, в обработчике было создано и показано окно информации. Наполнение окна реализуйте самостоятельно с учетом требований.

12) Теперь, когда все обработчики готовы, соедините их с нужными сигналами с помощью функции connect() в конструкторе главного окна:

```
connect(ui->Save, &QAction::triggered, this, &MainWindow::onSaveAction);
connect(ui->Show_history, &QAction::triggered, this, &MainWindow::onShowHistory);
connect(ui->Hide_history, &QAction::triggered, this, &MainWindow::onHideHistory);
connect(ui->About, &QAction::triggered, this, &MainWindow::onAbout);
```

Рисунок 14. Сигнально-слотовые соединения для пунктов меню

13) Добавьте недостающие горячие клавиши для управления меню и историей.

14) Протестируйте работу полученного приложения, проверив следующие случаи:

- При выборе в меню «История → Показать» размер главного окна расширяется и появляется виджет истории.

- При выборе «Меню → Сохранить значение» и пустом поле displayPanel в строке состояния появляется сообщение о том, что пустое значение не добавлено в историю.
- Сохранение нескольких чисел → история должна расти.
- Нажатие на элемент истории → значение появляется в displayPanel.
- Кнопка "Очистить" → список истории становится пустым.
- Кнопка "Скрыть" → окно истории исчезает, размер главного окна сужается до размера основного калькулятора.
- При повторном показе истории список значений актуальный.

Дополнительное задание: реализовать сохранение истории в файл, загрузку истории из файла.

Контрольные вопросы:

1. Как работает механизм сигналов и слотов?
2. Какие компоновщики (layout) используются в Qt и зачем они нужны?
3. Что такое QSplitter и в чём его отличие от других layout-компонентов?
4. Для чего используется statusBar в приложении? Приведите примеры использования.
5. Как работают горячие клавиши в Qt? Как их назначить для действий?
6. Как реализовать реакцию на клик по элементу списка (QListWidgetItem)?
7. Как сделать так, чтобы главное окно расширялось и сужалось при показе/скрытии боковой панели истории?
8. Как добавить новую форму в проект?
9. Поясните как связана работа с QAction и QMenu в данном проекте.
10. Когда используются сигналы triggered() и clicked() в Qt?