Практическое задание №3. Тема: Расширение списка тем оформления в приложении для работы с библиотекой подсветки QSourceHighlighter в среде QtCreator.

Цель работы: Получить практические навыки расширения функциональности приложения на основе библиотеки Qt, научиться работать с мета-объектной системой Qt.

Задание:

- Добавить в проект редактора с подсветкой кода QSourceHighlighter две новые темы оформления.
- Предусмотреть динамическую смену оформления, включая фон всего редактора в зависимости от выбранной темы.
- При сохранении документа в формате JSON сохранять также тему с фоном и восстанавливать ее при открытии файла.

Порядок выполнения:

1) Откройте файл qsourcehighliterthemes.cpp, реализацию функции theme() и обратите внимание, что там задана всего одна тема Monokai.

2) Откройте файл qsourcehighliter.h и добавьте в перечисление Themes новые темы, например темную и светлую:

```
enum Themes {
    Monokai = 1,
    DarkTheme = 2,
    LightTheme = 3,
};
```

3) Реализуйте эти темы в qsourcehighliterthemes.cpp аналогично функции monokai() . Данная функция возвращает ассоциативный массив (словарь) QHash<Token, QTextCharFormat>, который связывает токен (тип текста для форматирования) с его форматом. То есть для реализации темы необходимо определить цвета и стили для каждого типа токенов.

4) Добавьте вызов новых тем в метод theme().

5) Откройте файл mainwindow.cpp и добавьте новые темы для выбора в интерфейс, дописав метод initThemesComboBox(). Начальным значением по умолчанию в выпадающий список тем задайте значение «default», при выборе которого должен быть использован стандартный стиль текста Qt.

6) Для смены фона виджета редактора добавьте метод applyEditorBackground(), который будет вызываться из themeChanged() с выбранной из списка темой. Объявите его в заголовочном файле mainwindow.h следующим образом:

```
  void applyEditorBackground(QSourceHighlite::QSourceHighliter::Themes
theme);
```

Реализация метода смены фона редактора соответственно теме в файле mainwindow.cpp:

```
void MainWindow::applyEditorBackground(QSourceHighliter::Themes theme) {
 // Получаем форматы для текущей темы
 auto themeFormats = QSourceHighliterTheme::theme(theme);
 // Берём формат блока — он содержит фон всей строки
 QTextCharFormat blockFormat =
themeFormats.value(QSourceHighliter::CodeBlock);

  // Получаем цвет фона. Если его нет — используем белый
  QColor bgColor = blockFormat.background().color();
  if (!bgColor.isValid()) {
      bgColor = Qt::white;
  }
  // Устанавливаем стиль только для фона
  QString styleSheet = QString("QPlainTextEdit { background-color: %1; }").
arg(bgColor.name());
  // Применяем стиль ко всему редактору
  ui->plainTextEdit->setStyleSheet(styleSheet);
}
```

7) Доработайте сохранение файла в формате JSON, добавив тему оформления в файл. При открытии соответственно тема и фон должны быть восстановлены вместе с текстом и языком.

8) Протестируйте работу полученного приложения, проверив следующие случаи:

- При запуске приложения отображается стандартная цветовая схема оформления (тема «Default»).
- При открытии текстового файла тема остается текущей.
- При смене темы происходит обновление оформления текста и фона.
- Сохранение/открытие JSON → тема и язык должны сохраниться.

Дополнительное задание: реализовать диалог с возможностью создания пользовательской темы оформления. Тема должна сохраняться в отдельный файл и подгружаться в список тем при каждом запуске редактора.

Контрольные вопросы:

1. Как передается значение из QComboBox в слот выбора темы?
2. Какую роль играет QHash<QSourceHighliter::Token, QTextCharFormat> в реализации тем оформления?
3. Какие элементы кода окрашиваются в функции описания темы оформления? Перечислите все токены и их назначение.
4. Какой класс в Qt используется для задания форматирования текста, и как он применяется при реализации подсветки синтаксиса через QSourceHighliter?
5. Как сохранить текущую тему оформления в JSON-файле вместе с текстом и языком программирования?