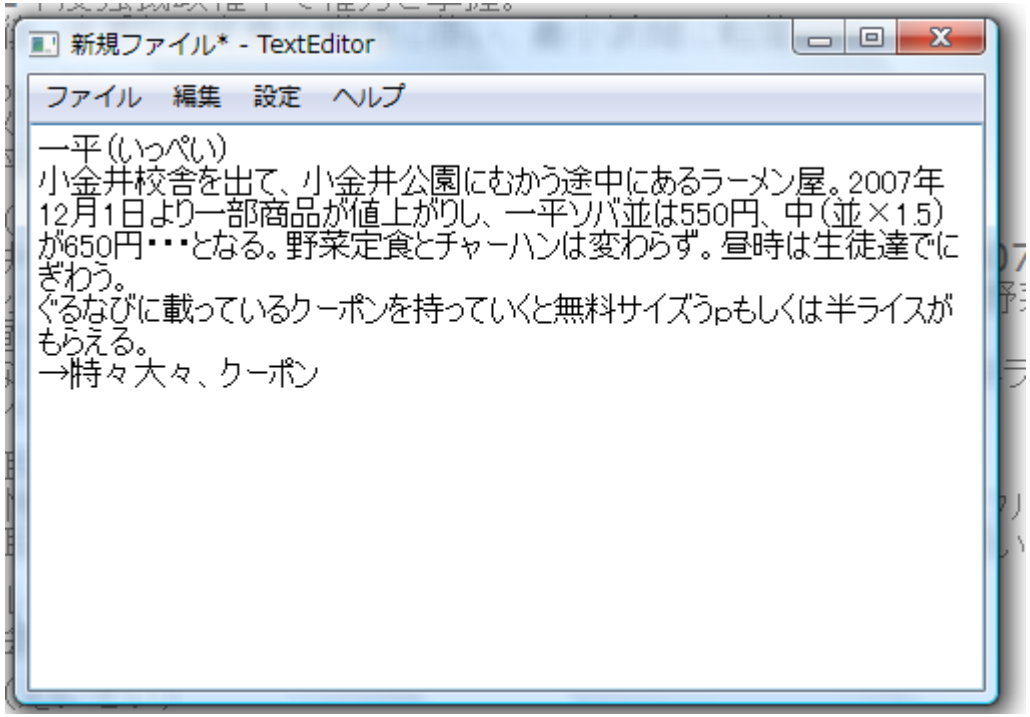
第三回
Qtの使い方講座

簡単なアプリを作ってみよう Part Ⅱ

- 今回の課題：QMainWindowクラスを拡張して、メモ帳アプリを作成する



メモ帳アプリの大まかな仕様

- テキストエリアで文字列を編集する
- 編集したテキストを保存する
- 既存のテキストファイルを開いて編集・保存
- ウィンドウを閉じる時に保存するかどうかをユーザに問いかける
- オプション
  - フォントを設定
  - 背景色を設定

必要となるクラス

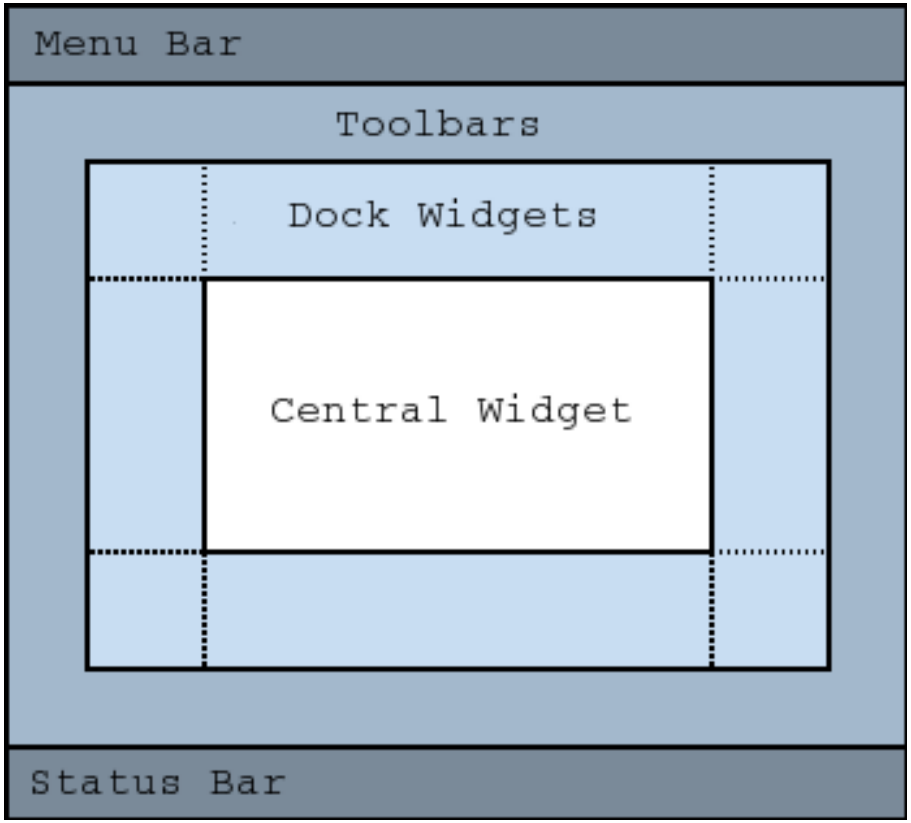
- QMainWindow
  …ベースとなるウィンドウ．QWidgetに様々な機能を拡張している
- QTextEdit　…複数行のテキスト編集
- QFile　…ファイルの読み書き
- QFileDialog　…ファイルダイアログを開く
- QAction
  …ユーザの行う操作を表し、メニュー等に組み込むことができる
- QMenu
  …MainWindowのメニューバーに追加して、ユーザに操作メニューを提供する
- QMessageBox　…メッセージダイアログを開く

QMainWindowクラス

- QWidgetの拡張クラス
- セントラルウィジェット、メニューバー、ツールバー、ステータスバー、ドックウィジェットの領域が予め用意されている
- セントラルウィジェットにメインとなるウィジェットを配置し、メニューバーに主要な操作をまとめるような組み方が基本となる

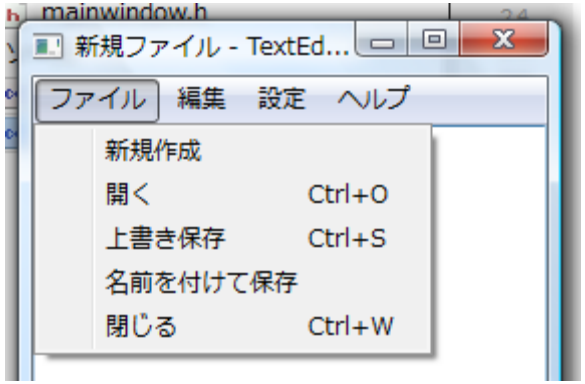
QMainWindowクラス

イメージ（拾い物）



QMainWindowクラス

- メニューの追加
  - QMainWindow::menuBar()->addMenu(“メニュー名”)で返ってきたポインタをQMenuポインタのインスタンスに格納する



- セントラルウィジェットの配置
  - QMainWindow::setCentralWidget(Qwidget *widget)

QActionクラス

- QObject::connect 関数で実行する処理を設定できる
  逆に言えば、これをやらないと全く意味を持たない
- アクションはメニューバー、ツールバー、コンテキストメニュー（右クリックメニュー）等に表示させられる
- 一つのアクションを複数の場所に配置できる
- ショートカットを設定できる

QActionクラス

- 初期化例

```
QAction *closeAction ;      //ヘッダファイル
closeAction = new Qaction(“閉じる”, this);
//ソースファイル（thisはMainWindowインスタンス）
```

- スロット接続例

```
connect( closeAction, SIGNAL( triggered() ),
                this, SLOT( close() ) );
```

QFileクラス

- 標準のFILE型と使い方はほとんど同じ
- 使用例（読み込み）

```
QFile file(fileName);          //ファイル名を設定
if (!file.open(QIODevice::ReadOnly))   //読み込み専用でオープン
        hogehogeError(); //エラー処理
QTextStream in(&file);       //テキストストリームにファイルをセット
QString str = in.readAll();   //テキストストリームから文字列を読み込む
```

QFileクラス

- 標準のFILE型と使い方はほとんど同じ
- 使用例（書き込み）

```
QFile file(fileName);          //ファイル名を設定
if (!file.open(QIODevice::WriteOnly))   //読み込み専用でオープン
        hogehogeError(); //エラー処理
QTextStream out(&file);     //テキストストリームにファイルをセット
QString str  =“hogehoge”;
out << str;   //テキストストリームを介して文字列をファイルに書き込む
```

QFileDialogクラス

- ファイルダイアログを開いてくれるクラス
- ファイルオープンダイアログ
  - QFileDialog::getOpenFileName()
  - 引数は　親ウィジェット、タイトル、ファイル名、拡張子フィルタ
  - 例：QFileDialog::getSaveFileName(this, "テキストの保存", ".", "テキストファイル (*.txt)");
- ファイルセーブダイアログ
  - QFileDialog::getSaveFileName()
  - 引数は上に同じ
- どちらのダイアログもQString型を返す

ウィンドウを閉じる時の処理

- ファイル編集を行うアプリに求められる挙動
  - ウィンドウを閉じる時に編集中のファイルを保存するか否かをユーザーに問いかける
  - この動作がないとソウルジェムが真っ黒に…

あたしってほんとバカ…

ウィンドウを閉じる時の処理

- クローズイベントをオーバーライド
  - ウィンドウを閉じる時の処理を上書きする
  - ここでメッセージダイアログを表示して、ユーザに選択肢を示す

```
void MainWindow::closeEvent(QCloseEvent *event)
{
   if(okToContinue()) {  //ユーザにメッセージを表示
      event->accept();  //イベントを受理する（ここではクローズイベントのこと）
   } else {
      event->ignore();  イベントを無視する（ウィンドウは閉じない）
   }
}
```

ウィンドウを閉じる時の処理

- ユーザに問いかけを行うokToContinue関数の実装

```
bool MainWindow::okToContinue()
{
   if(isWindowModified()) {    //ウィンドウの”変更有り”のフラグを見る
      int ret = QMessageBox::warning(this, tr("TextEditor"),
                                     "テキストは変更されています。¥n変更を保存しますか？",
                                     QMessageBox::Yes | QMessageBox::Default,
                                     QMessageBox::No,
                                     QMessageBox::Cancel | QMessageBox::Escape);
      if(ret == QMessageBox::Yes)
         return save();              //ファイル保存処理
      else if(ret == QMessageBox::Cancel)
         return false;
   }
   return true;
}
```

- Yes：ファイルの保存、No：保存せずに閉じる、Cancel：保存せずウィンドウを閉じない

ウィンドウを閉じる時の処理

- ウィンドウには内容が変更されたかどうかを示す modified というフラグがある
- setModified(bool) で書き換え可
- isModified() でフラグを取得

- 今回の場合は、テキストが変更された時にsetModified(true)を実行すればよい
- 保存やオープンした時にはfalseを投げる