Leow Wee Kheng
CS3249 User Interface Development
Department of Computer Science, SoC, NUS

Main Window

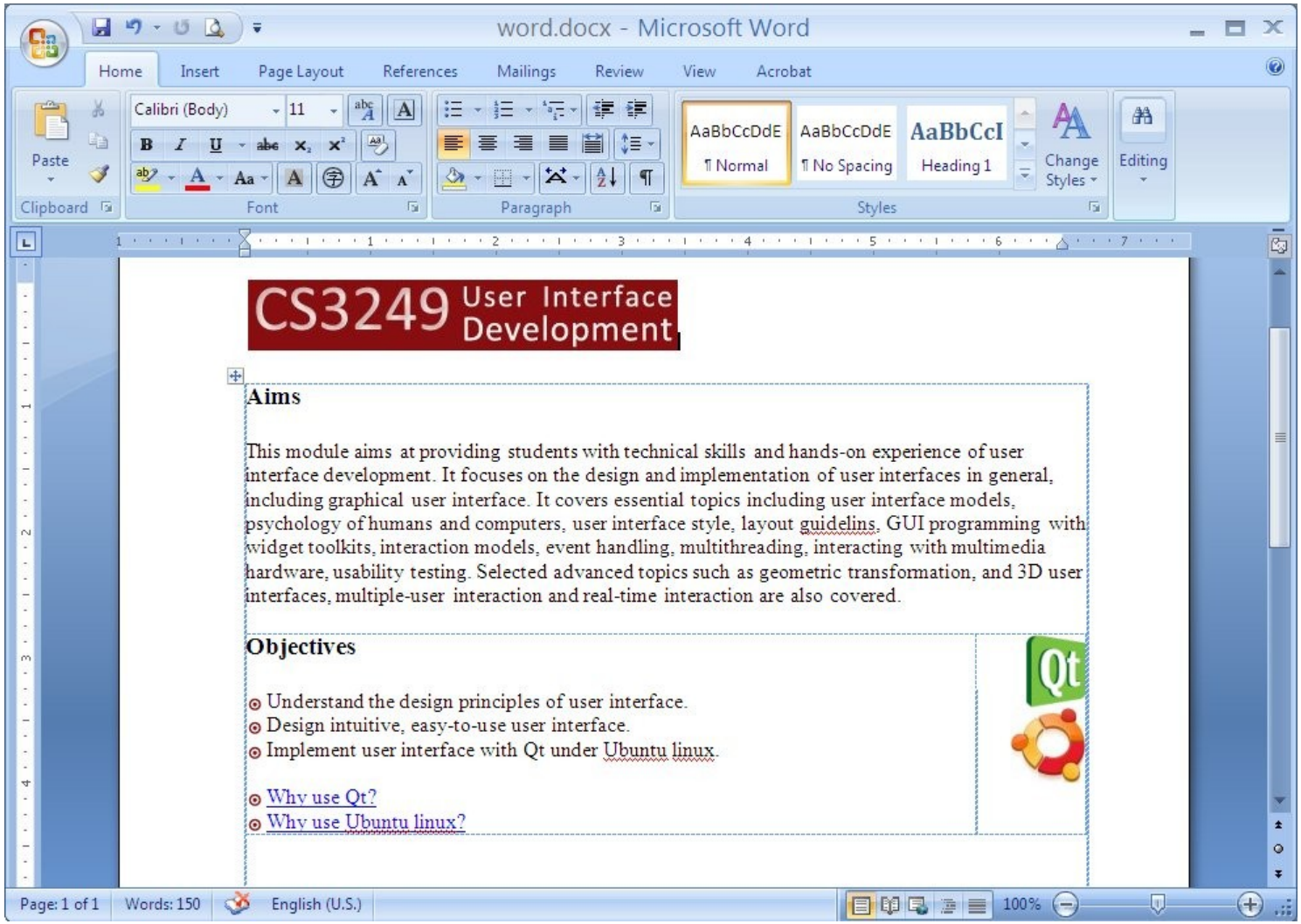

Main window works in opposite direction.

Most applications have a main window...

Mozilla Firefox web browser

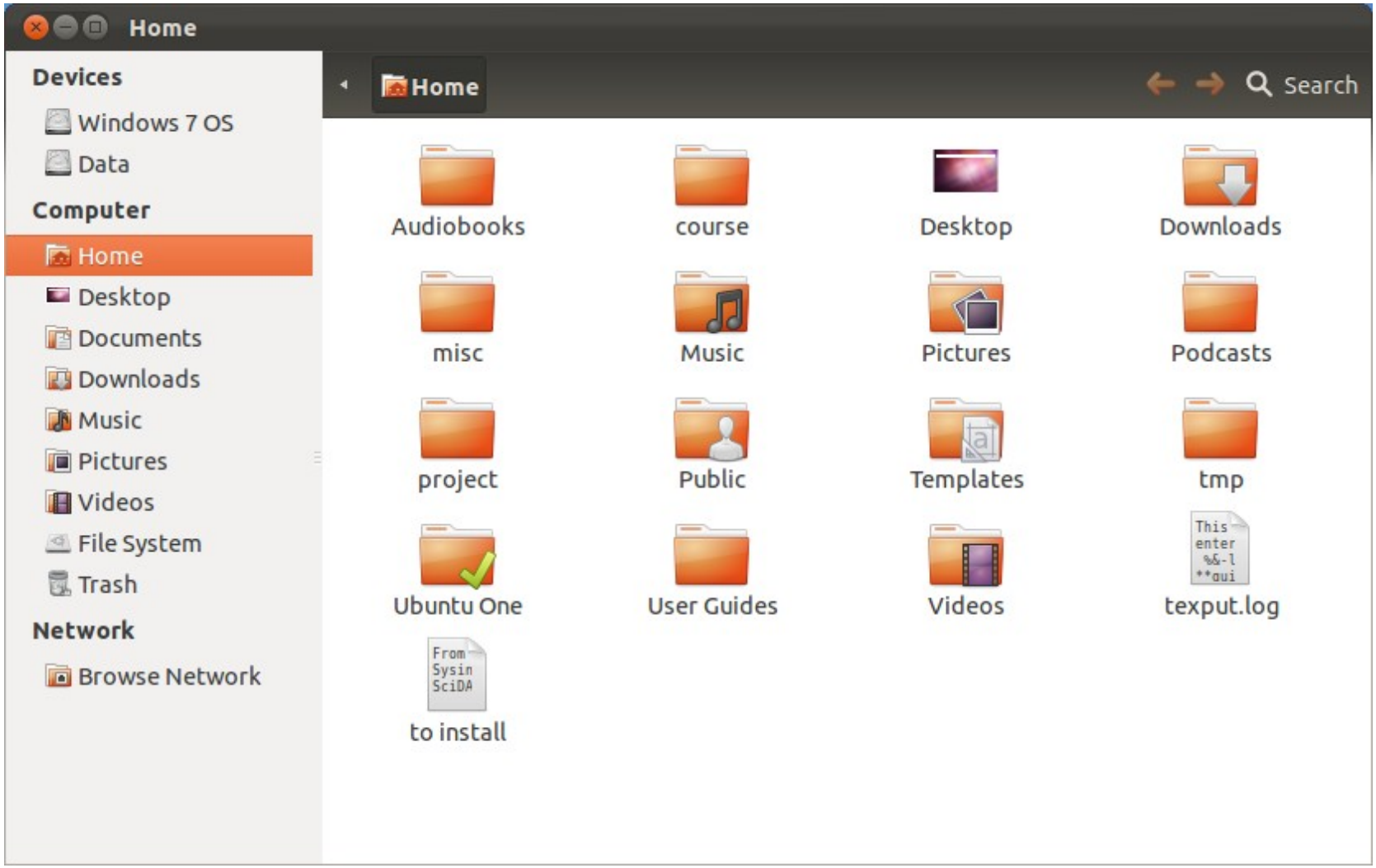
Most applications have a main window...

Microsoft Word



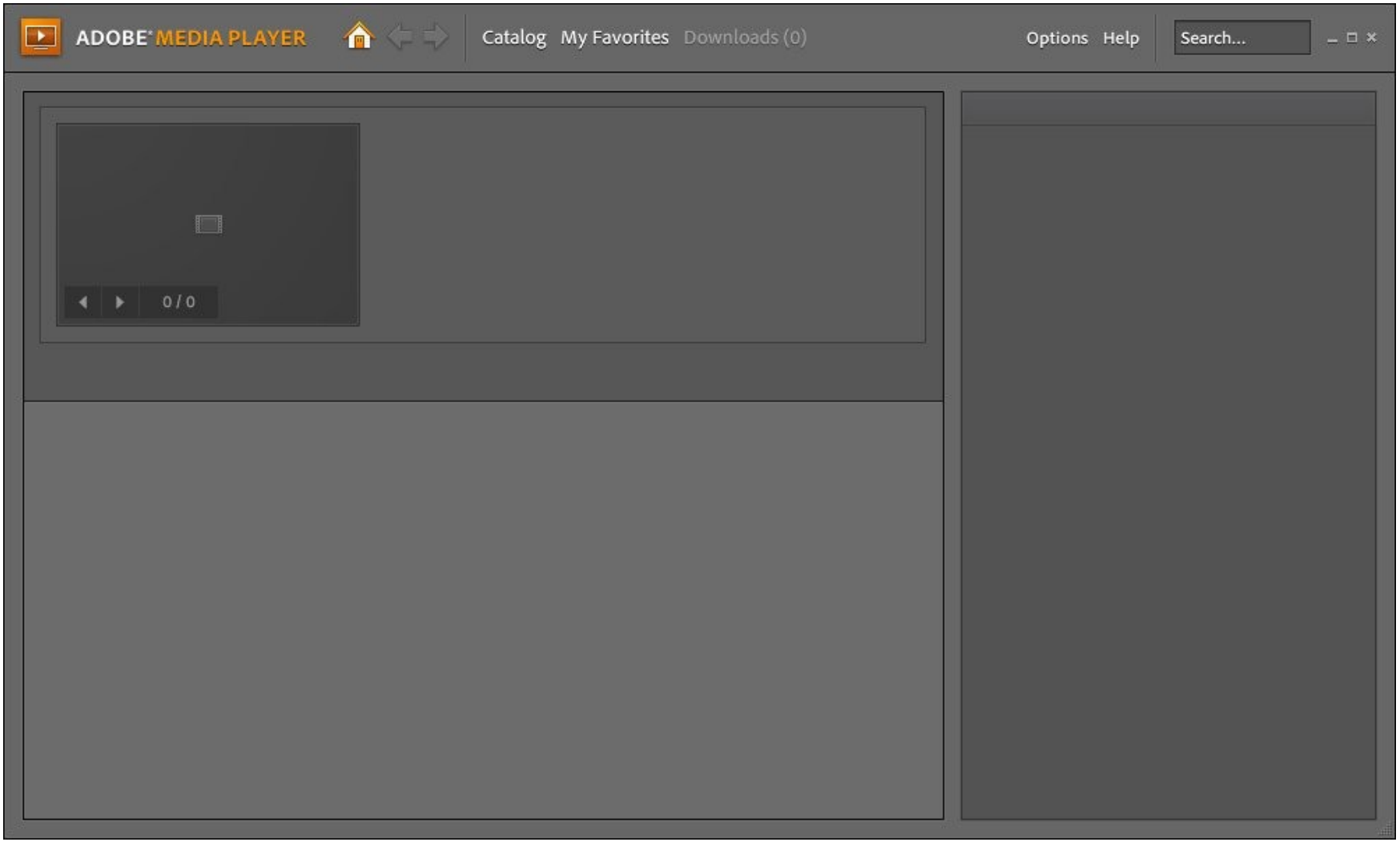
Most applications have a main window...

Nautilus file manager



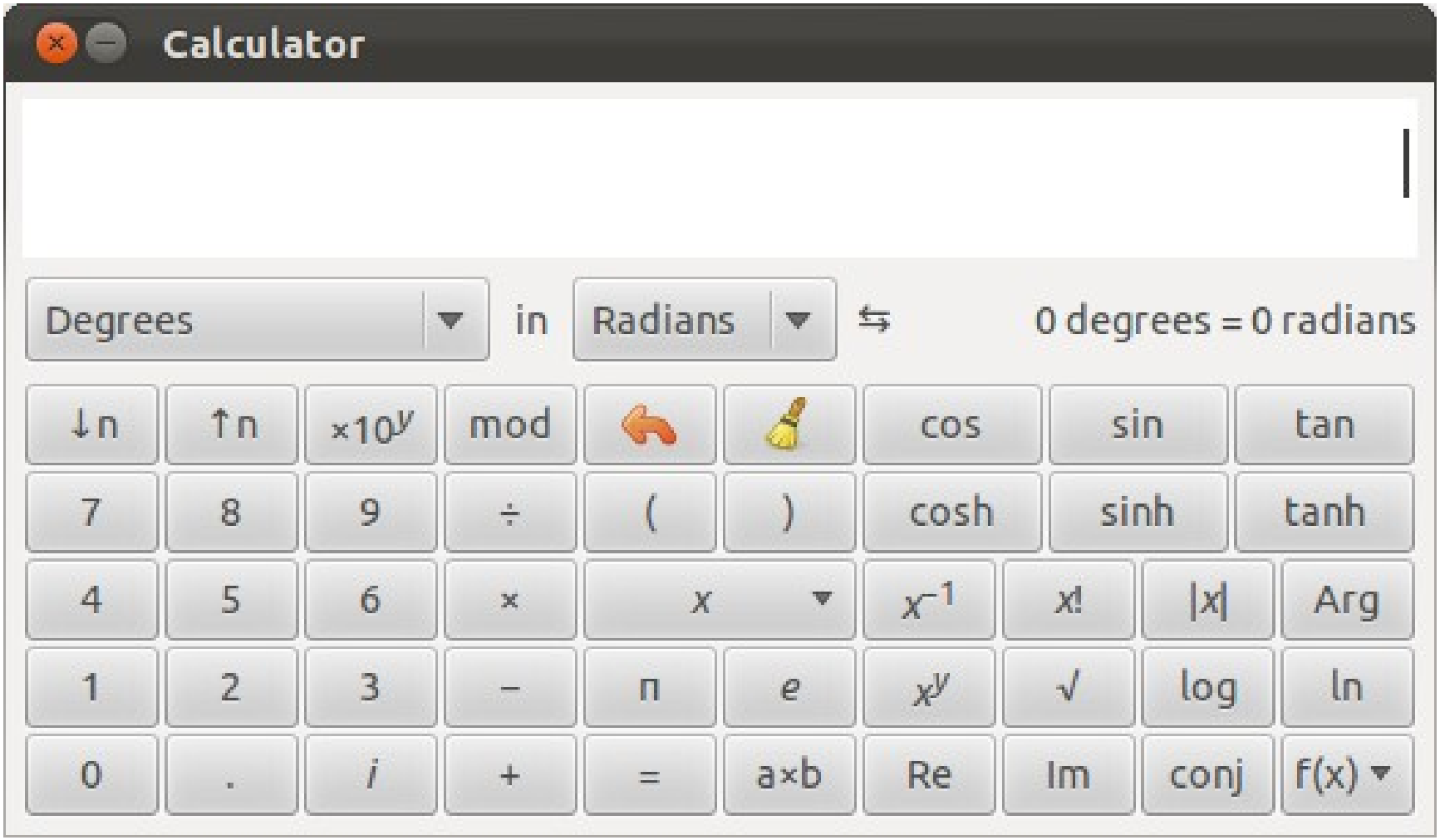
Most applications have a main window...

Adobe media player



Most applications have a main window...

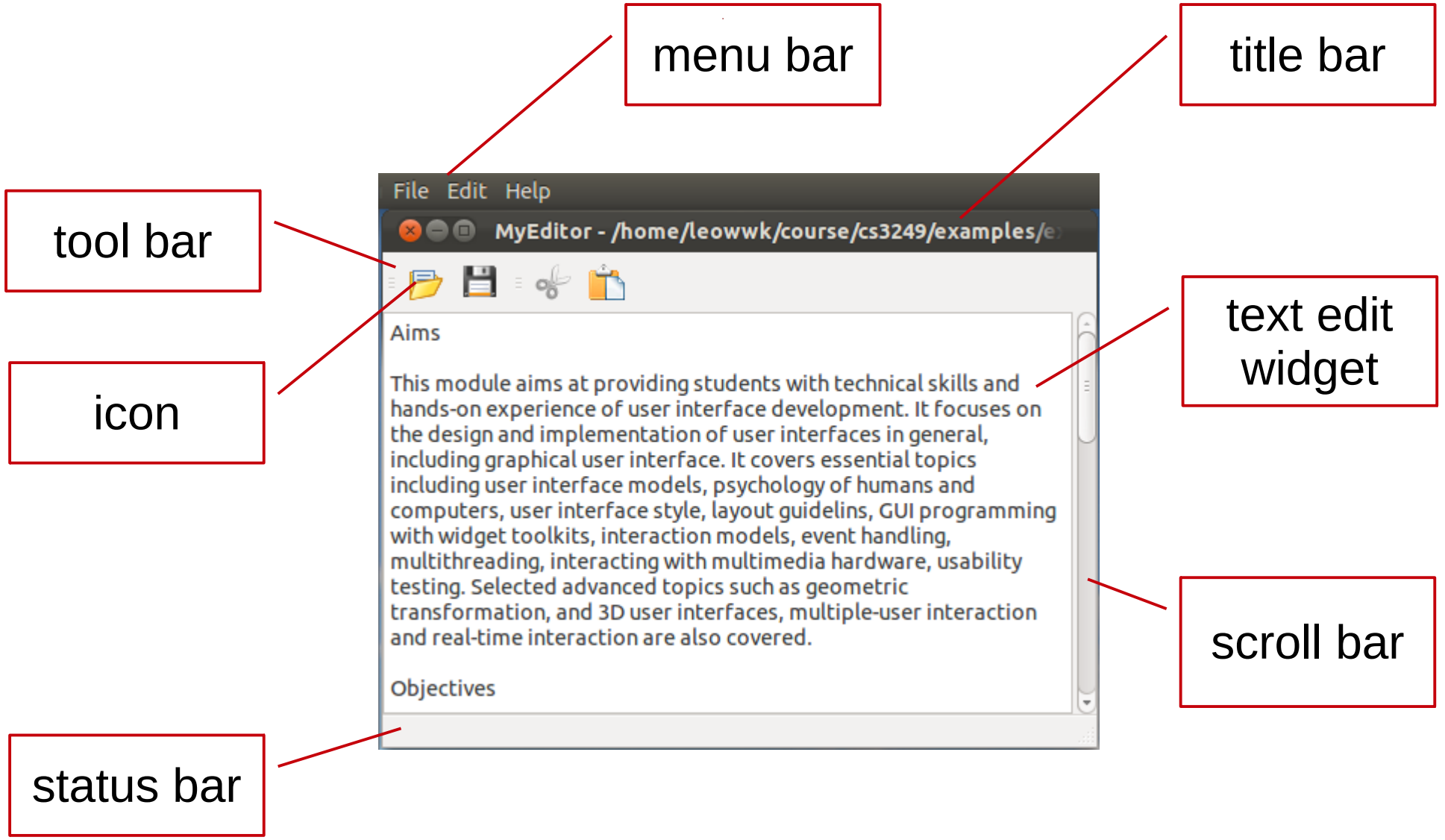
Calculator



Main Window

- Qt's main window widget `QMainWindow` comes with
  - menu bar, tool bar, status bar, scroll bars, etc.
- Convenient for building applications.
- Illustrate using `MyEditor`, a simple editor.

MyEditor: A Simple Editor



- Create `MyEditor` as subclass of `QMainWindow`.
- 4 main files:
  - resource file: specifies icon images
  - header file of main window: defines object class
  - program file of main window: defines class functions
  - main program file: puts everything together

Resource file `myeditor.qrc`

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
    <file>images/cut.png</file>
    <file>images/myeditor.png</file>
    <file>images/open.png</file>
    <file>images/paste.png</file>
    <file>images/save.png</file>
</qresource>
</RCC>
```

```
// MyEditor.h
// Simple text editor.

#ifndef MYEDITOR_H
#define MYEDITOR_H

#include <QMainWindow>

// Class declaration
class QAction;
class QMenu;
class QPlainTextEdit;

class MyEditor: public QMainWindow  // inherit QMainWindow
{
    Q_OBJECT  // macro for meta object features

public:
    MyEditor();
```

```
protected:
    void closeEvent(QCloseEvent *event);

private slots:
    void open();
    bool save();
    void about();
    void isModified();

private:
    void createWidgets();
    void createActions();
    void createMenus();
    void createToolBars();
    void createStatusBar();
```

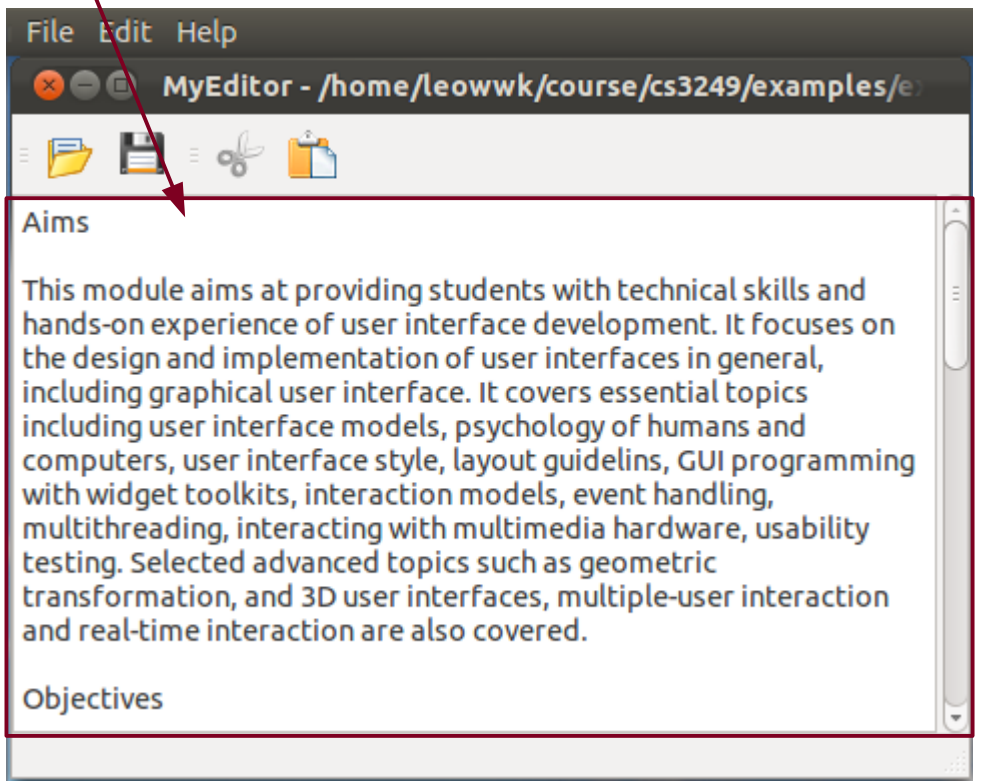
```
private:
    QAction *openAction;  // actions for menu item
    QAction *saveAction;
    QAction *exitAction;
    QAction *cutAction;
    QAction *pasteAction;
    QAction *aboutAction;

    QMenu *fileMenu;
    QMenu *editMenu;
    QMenu *helpMenu;

    QToolBar *fileToolBar;
    QToolBar *editToolBar;
```

```
private:
    // Widgets and variables
    QPlainTextEdit *textEdit;  // text editor widget
    QString currFile;

    // Supporting methods
    bool okToContinue();
    void loadFile(const QString &fileName);
    bool saveFile(const QString &fileName);
    void setCurrentFile(const QString &fileName);
};

#endif
```

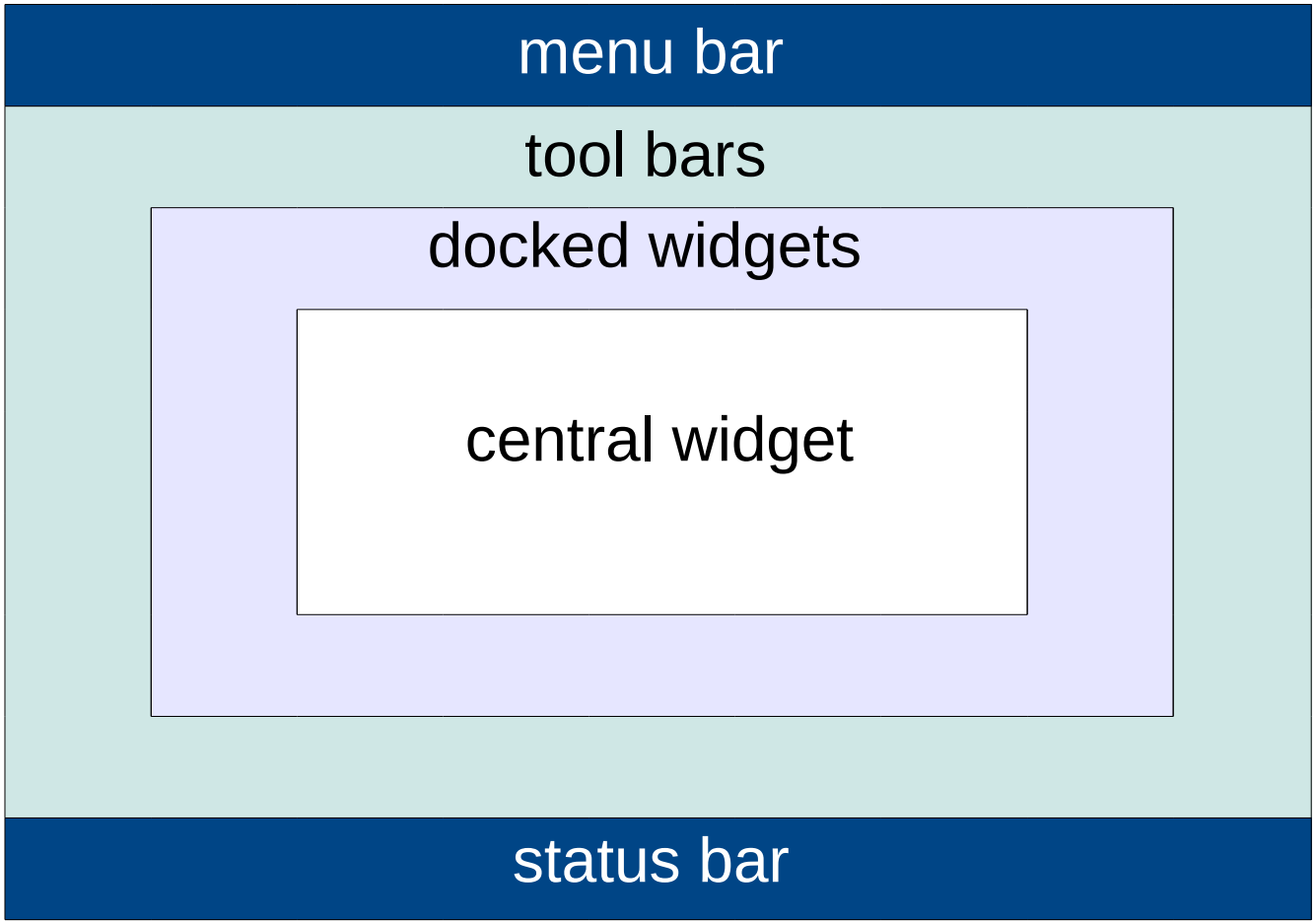
```
// MyEditor.cpp
// Simple text editor.

#include <QtGui>
#include "MyEditor.h"


// Constructor

MyEditor::MyEditor()
{
    createWidgets();
    createActions();
    createMenus();
    createToolBars();
    createStatusBar();
}
```

```
void MyEditor::closeEvent(QCloseEvent *event)
{
    if (okToContinue())
        event->accept();  // auto close child widgets
    else
        event->ignore();
}
```

```
void MyEditor::createWidgets()
{
    textEdit = new QPlainTextEdit;
    setCentralWidget(textEdit);  // Also set it as child.
    setWindowIcon(QIcon(":/images/myeditor.png"));

    // Initialisation
    SetGeometry(0, 0, 550, 650);  // location, size
    setCurrentFile("");
}
```

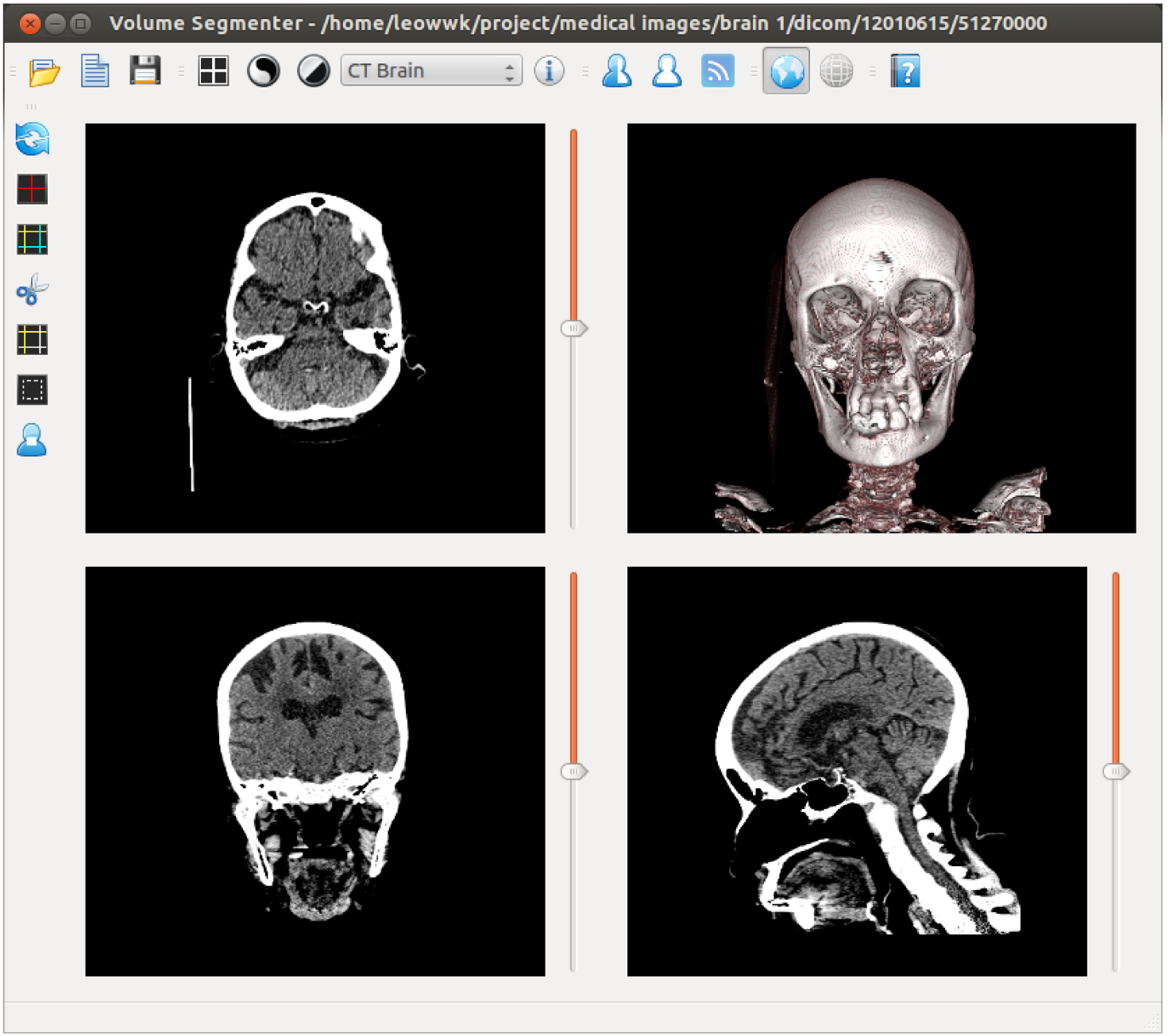


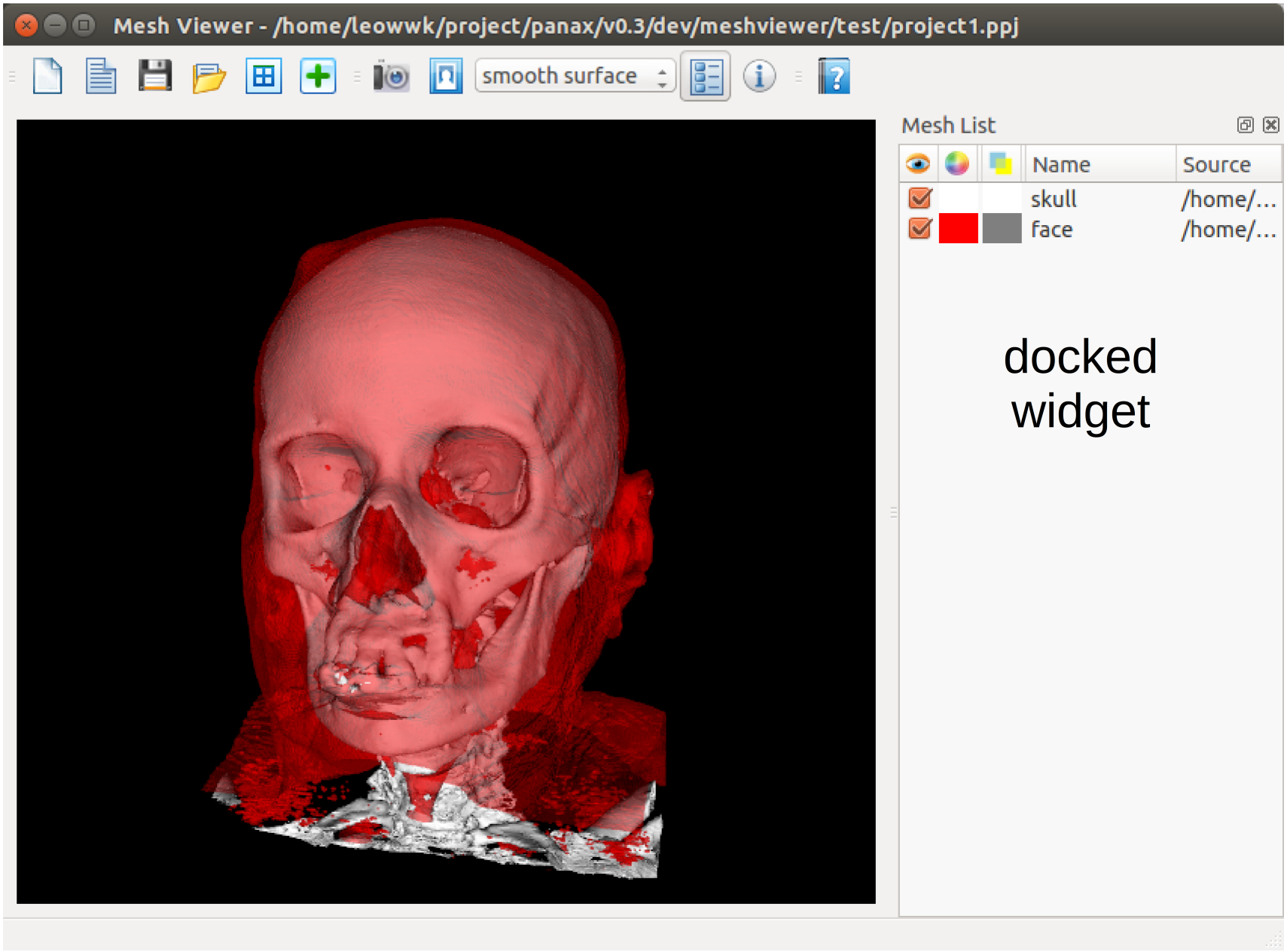
QMainWindow's layout

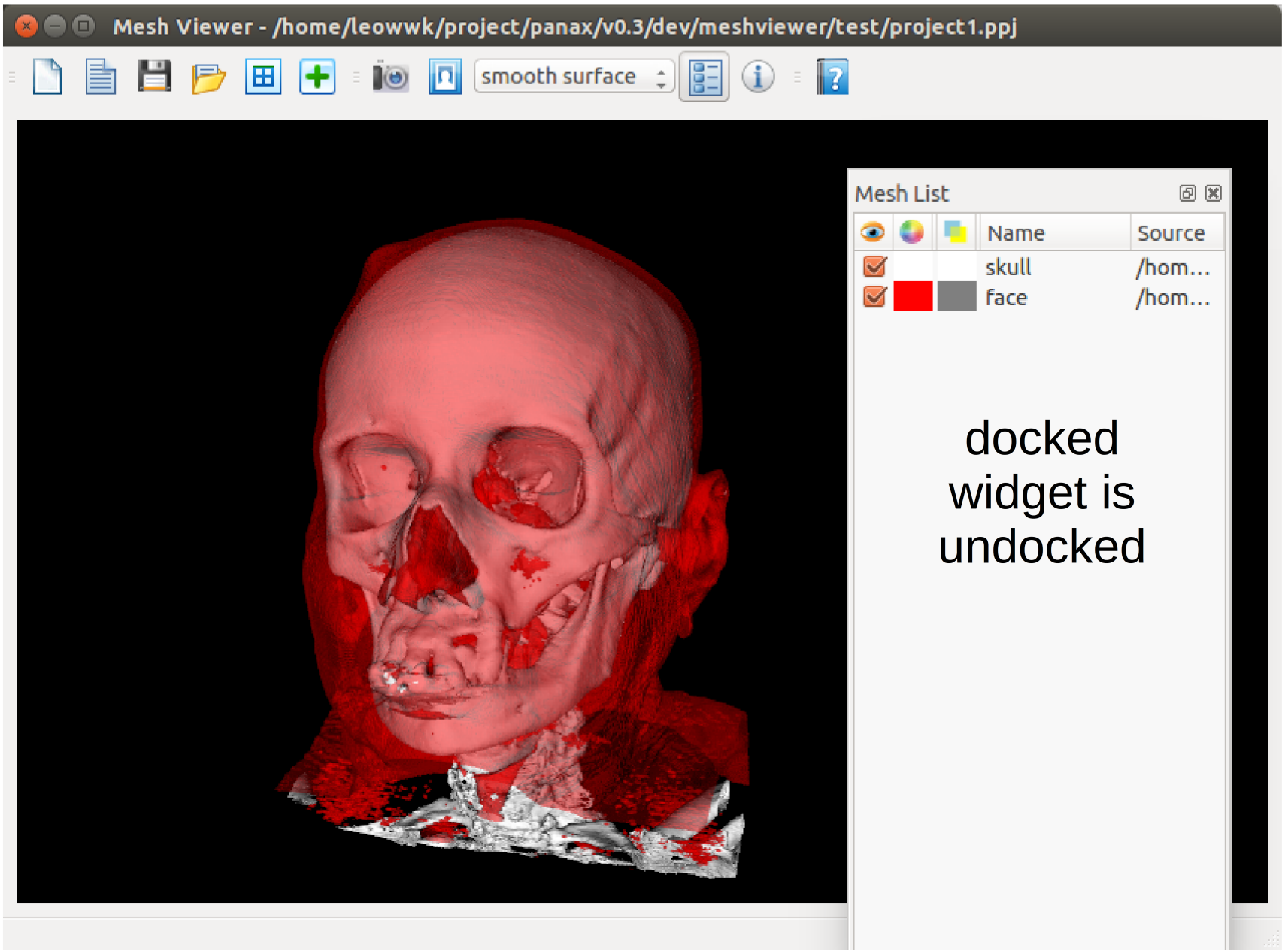


top tool bar

left side tool bar




Mesh Viewer - /home/leowwk/project/panax/v0.3/dev/meshviewer/test/project1.ppj
smooth surface
Mesh List
Name
Source
skull
/home/...
face
/home/...
docked
widget

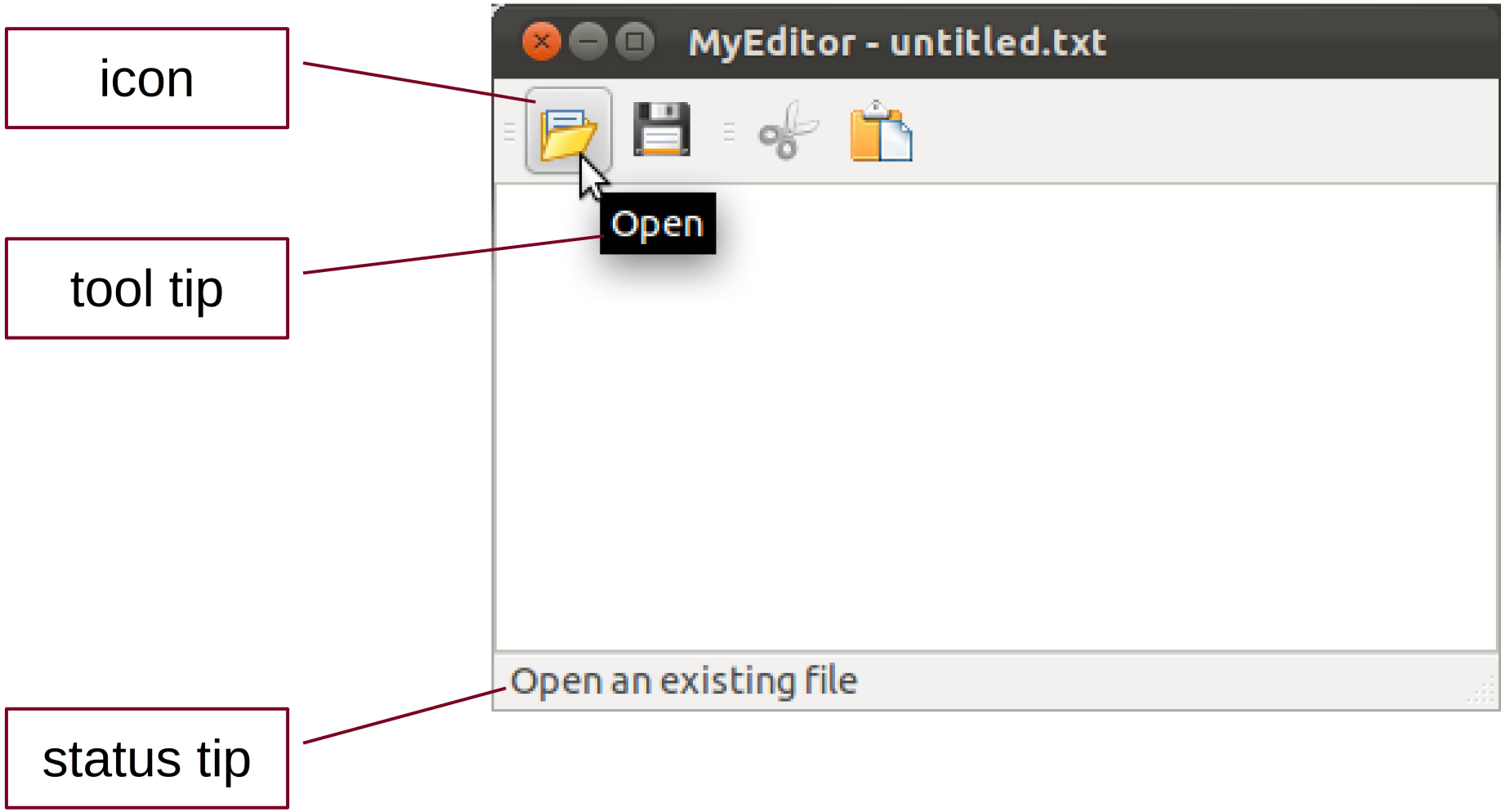

Mesh Viewer - /home/leowwk/project/panax/v0.3/dev/meshviewer/test/project1.ppj
smooth surface
Mesh List
Name
Source
skull
/hom...
face
/hom...
docked
widget is
undocked

QAction

- Action corresponding to user's command.
- Can be inserted into menus and tool bars.
- Contains
  - `text`: descriptive text for display in menu
  - `icon`: icon for display in tool bar
  - `shortcut`: keyboard short cut key sequence
  - `toolTip`: pop-up help text, default: `text`
  - `statusTip:` text for display in status bar
  - `data`: for storing data associated with action
  - etc.

```
void MyEditor::createActions()
{
    openAction = new QAction(tr("&Open"), this);
    openAction->setIcon(QIcon(":/images/open.png"));
    openAction->setShortcut(tr("Ctrl+O"));
    openAction->setStatusTip(tr("Open an existing file"));
    connect(openAction, SIGNAL(triggered()),
        this, SLOT(open()));
```

- Create action for opening a file.
- `tr` : translation for internationalisation.
- `&Open` : keyboard access of menu item: `Alt+O`.
- `this` is passed as the parent `QObject`.
- `Ctrl+O` : keyboard short cut.
- `connect` : connect `openAction` to `MyEditor`'s `open()`.


icon
tool tip
status tip
MyEditor - untitled.txt
Open
Open an existing file

```
saveAction = new QAction(tr("&Save"), this);
saveAction->setIcon(QIcon(":/images/save.png"));
saveAction->setShortcut(tr("Ctrl+S"));
saveAction->setStatusTip(
    tr("Save the document to a file"));
connect(saveAction, SIGNAL(triggered()),
    this, SLOT(save()));

exitAction = new QAction(tr("E&xit"), this);
exitAction->setShortcut(tr("Ctrl+Q"));
exitAction->setStatusTip(tr("Exit this MyEditor"));
connect(exitAction, SIGNAL(triggered()),
    this, SLOT(close()));
```

- `exitAction` is connected to built-in `close` slot.

```
cutAction = new QAction(tr("C&ut"), this);
cutAction->setIcon(QIcon(":/images/cut.png"));
cutAction->setShortcut(tr("Ctrl+X"));
cutAction->setStatusTip(
    tr("Cut selected contents to clipboard"));
connect(cutAction, SIGNAL(triggered()),
    textEdit, SLOT(cut()));
```

- `cutAction` is connected to `textEdit`'s built-in `cut()`.

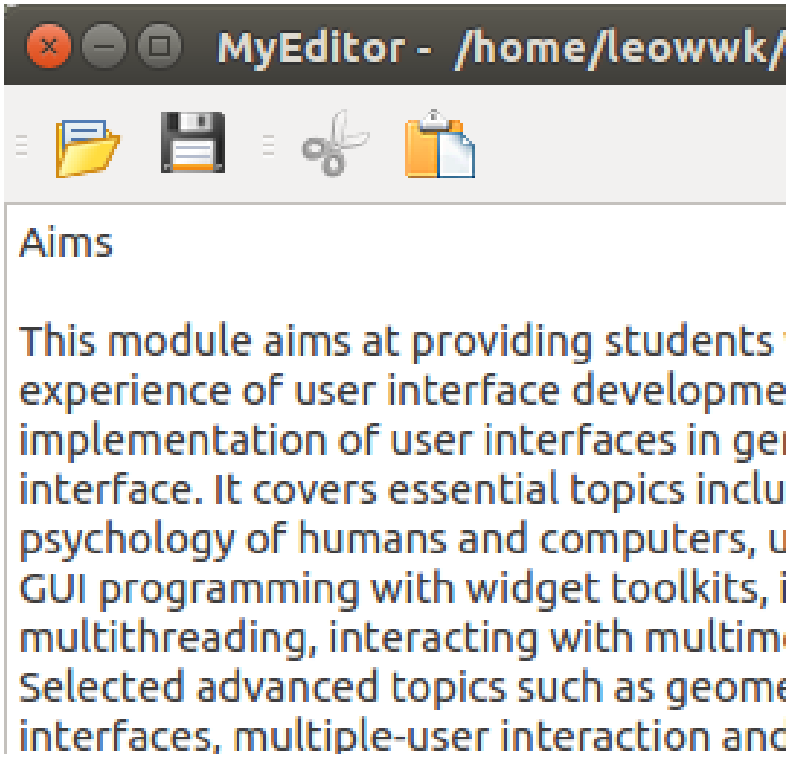
```
pasteAction = new QAction(tr("&Paste"), this);
pasteAction->setIcon(":/images/paste.png"));
pasteAction->setShortcut(tr("Ctrl+V"));
pasteAction->setStatusTip(
    tr("Paste clipboard's contents into current"
       "selection"));
connect(pasteAction, SIGNAL(triggered()),
    textEdit, SLOT(paste()));
```

- Strings in `tr()` are concatenated; no need ','
- `pasteAction` is connected to `textEdit`'s built-in `paste()`.

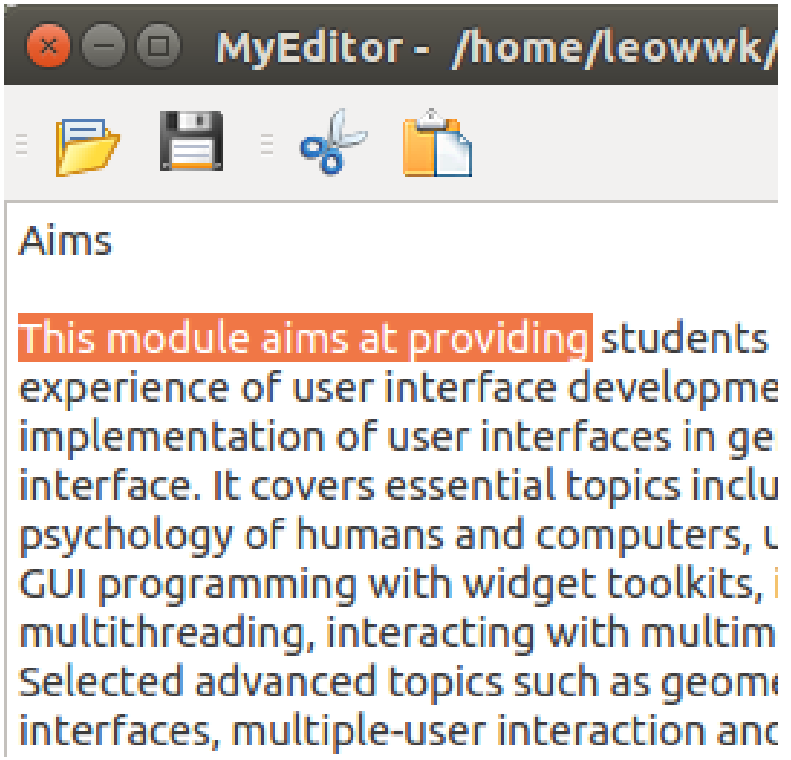
```
aboutAction = new QAction(tr("&About"), this);
aboutAction->setStatusTip(
    tr("Show information about this MyEditor"));
connect(aboutAction, SIGNAL(triggered()),
    this, SLOT(about()));
```

```
// Other connections
cutAction->setEnabled(false);
connect(textEdit, SIGNAL(copyAvailable(bool)),
        cutAction, SLOT(setEnabled(bool)));
```

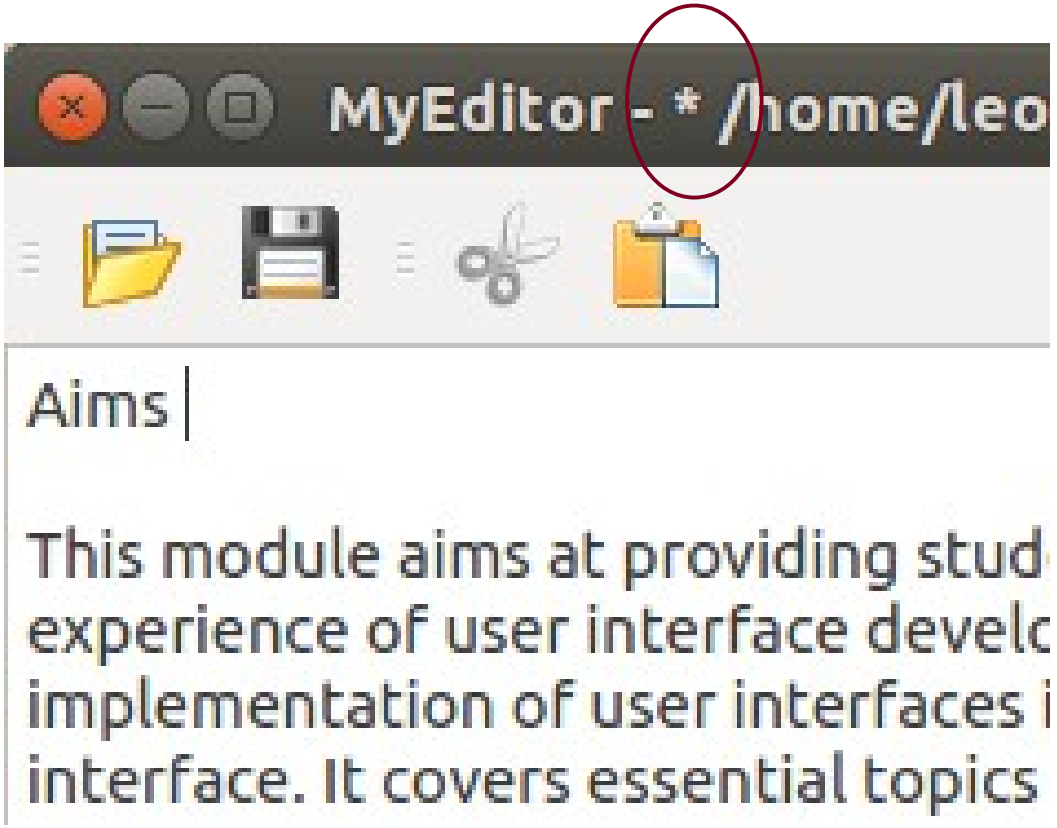
cut disabled



cut enabled

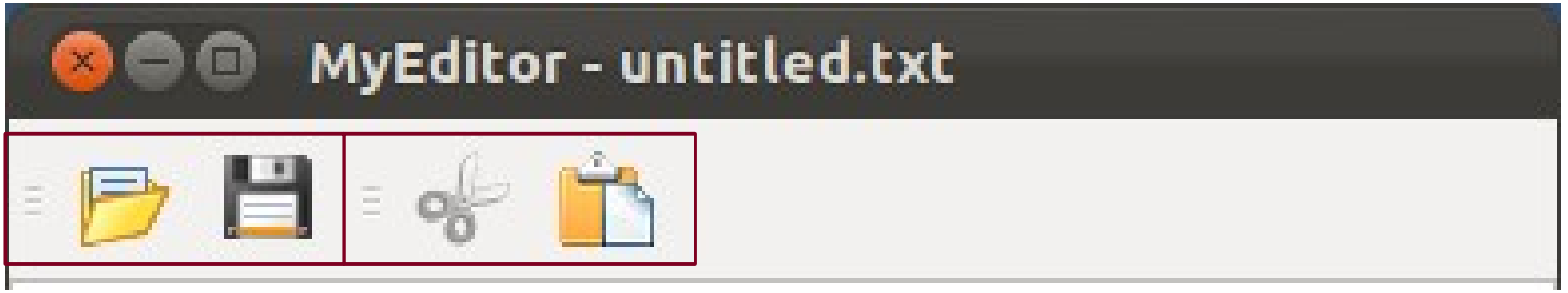


```
    connect(textEdit->document(),SIGNAL(contentsChanged()),
            this, SLOT(isModified()));
}

void MyEditor::isModified()
{
    setWindowModified(textEdit->document()->isModified());
}
```



```
void MyEditor::createMenus()
{
    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(openAction);
    fileMenu->addAction(saveAction);

    fileMenu->addSeparator();
    fileMenu->addAction(exitAction);

    editMenu = menuBar()->addMenu(tr("&Edit"));
    editMenu->addAction(cutAction);
    editMenu->addAction(pasteAction);

    menuBar()->addSeparator();

    helpMenu = menuBar()->addMenu(tr("&Help"));
    helpMenu->addAction(aboutAction);
}
```

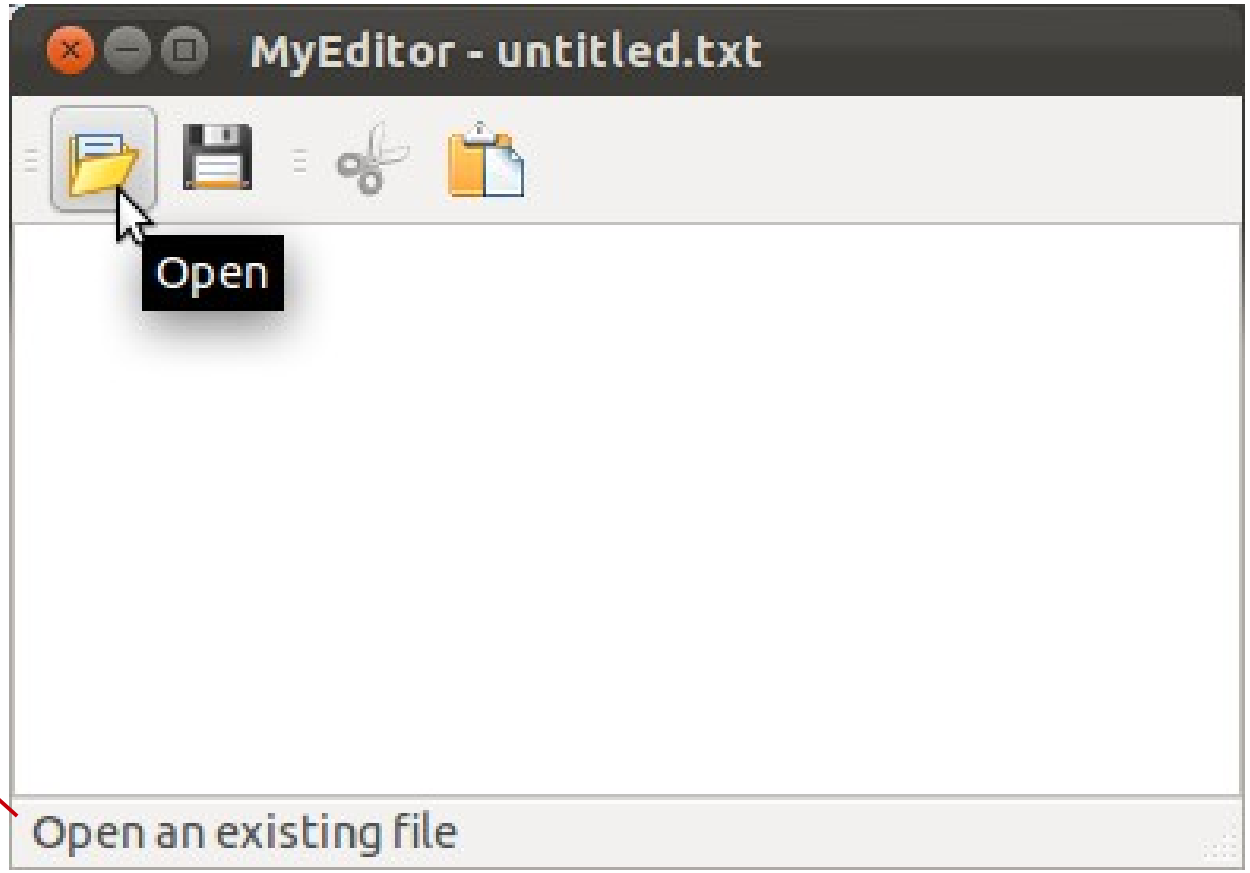
```
void MyEditor::createToolBars()
{
    fileToolBar = addToolBar(tr("File"));
    fileToolBar->addAction(openAction);
    fileToolBar->addAction(saveAction);

    editToolBar = addToolBar(tr("Edit"));
    editToolBar->addAction(cutAction);
    editToolBar->addAction(pasteAction);
}
```



fileToolBar editToolBar

```
void MyEditor::createStatusBar()
{
    statusBar()->showMessage(tr(""));
}
```

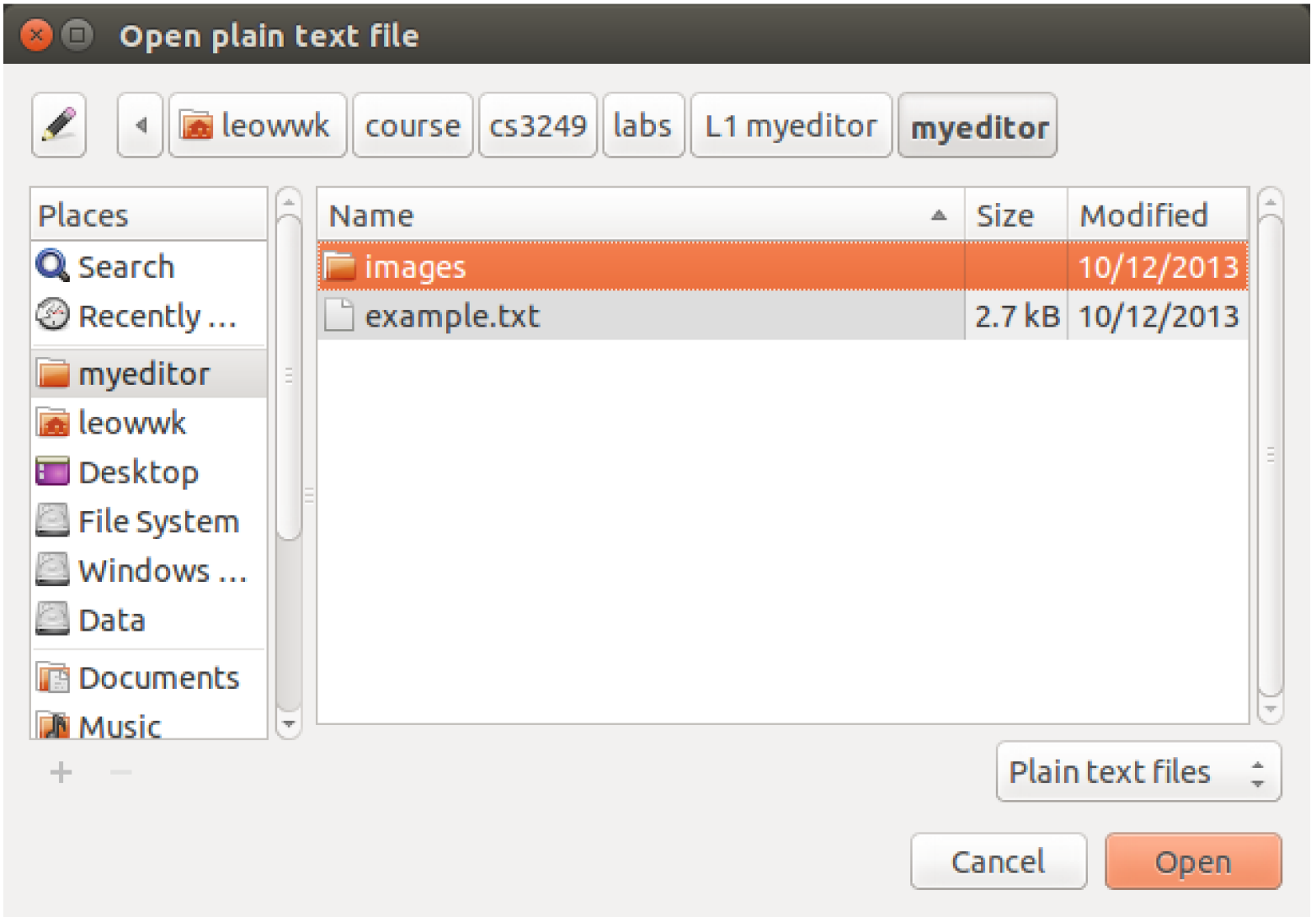
- Use default status bar; no need to create new one.



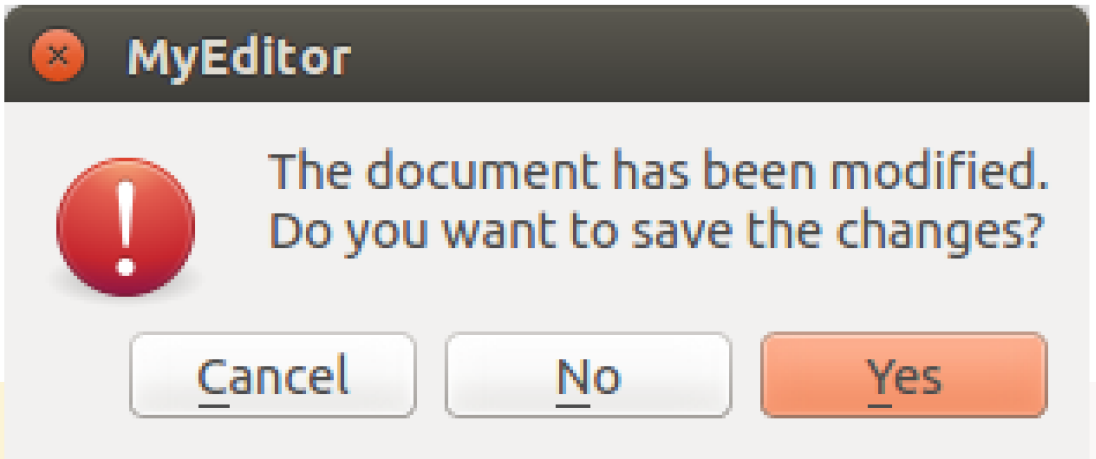
status bar

```
void MyEditor::open()
{
    if (okToContinue())
    {
        QString fileName =
            QFileDialog::getOpenFileName(
            this,                                   // parent widget
            tr("Open plain text file"),             // descriptive caption
            ".",                                    // working directory
            tr("Plain text files (*.txt);;          // file filter
                Any image files (*)"));

        if (!fileName.isEmpty())
            loadFile(fileName);
    }
}
```

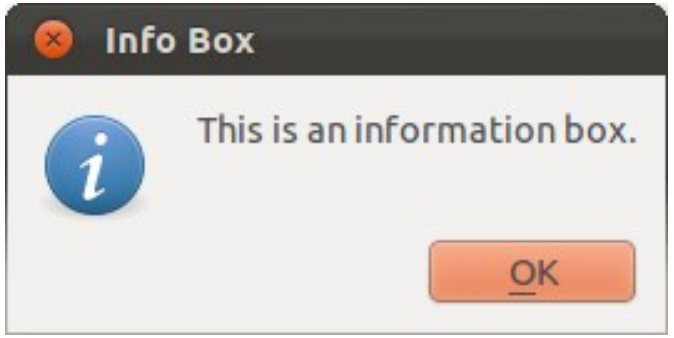
open file dialog

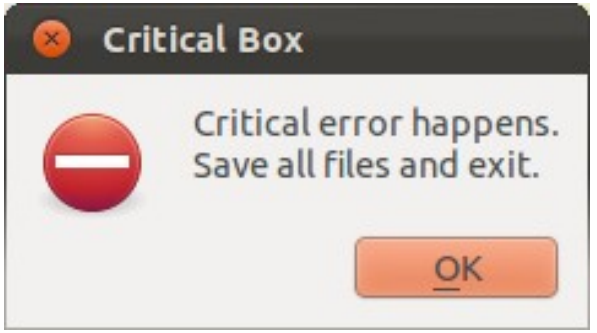


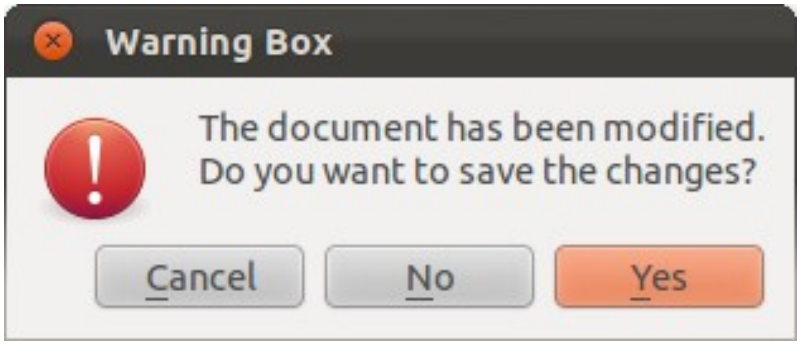
```
bool MyEditor::okToContinue()
{
    if (!(textEdit->document()->isModified()))
        return true;

    int code = QMessageBox::warning(this, tr("MyEditor"),
        tr("The document has been modified.\n"
           "Do you want to save the changes?"),
        QMessageBox::Yes | QMessageBox::Default,
        QMessageBox::No,
        QMessageBox::Cancel | QMessageBox::Escape);

    if (code == QMessageBox::Yes)
        return save();
    else if (code == QMessageBox::Cancel)
        return false;

    return true;
}
```

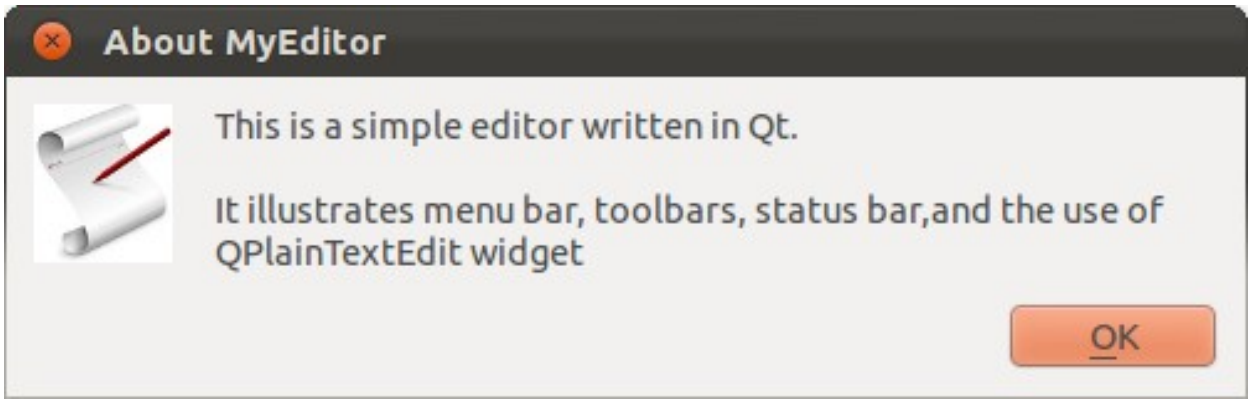


- `QMessageBox` supports 5 types of message box.
  - Default button is highlighted.





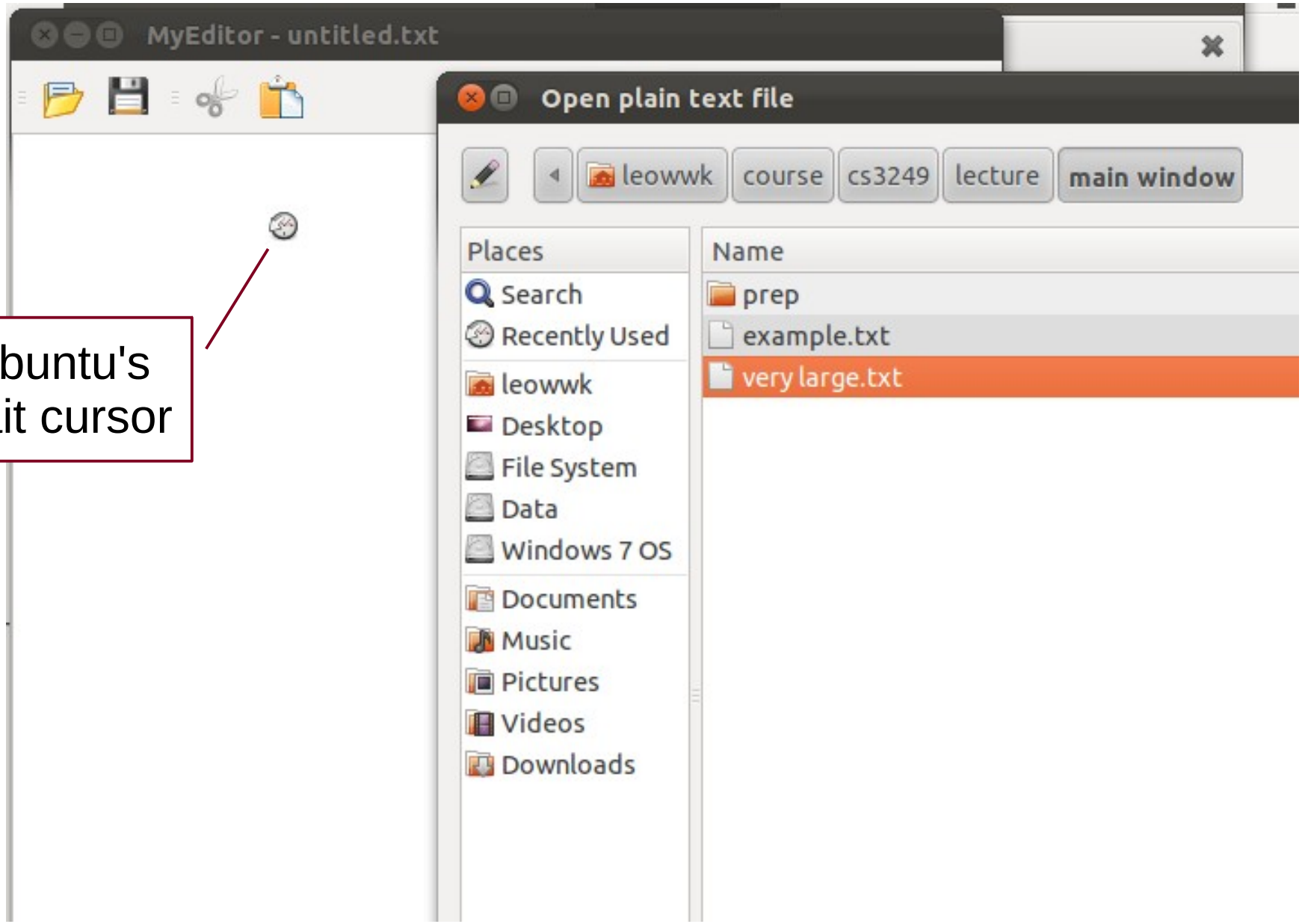




```
void MyEditor::loadFile(const QString &fileName)
{
    QFile file(fileName);
    if (!file.open(QFile::ReadOnly | QFile::Text))
    {
        QMessageBox::warning(this, tr("MyEditor"),
            tr("Cannot read file %1:\n%2.")
            .arg(fileName).arg(file.errorString()));
        return;
    }

    QTextStream in(&file);
    QApplication::setOverrideCursor(Qt::WaitCursor);
    textEdit->setPlainText(in.readAll());
    QApplication::restoreOverrideCursor();

    setCurrentFile(fileName);
    statusBar()->showMessage(tr("File loaded"), 2000);
}
```

duration
in milliseconds

Reading very large file...



QString

- Built-in string class
- String creation and concatenation

```
QString str1 = QString();  // empty QString
QString str2 = QString(“I like Qt”);
QString str3 = QString(“3249”);
QString str4 = str2 + “ and CS” + str3;
```

- Conversions

```
char *cstr = str4.toAscii().data();
int i = str3.toInt();
```

- Comparisons

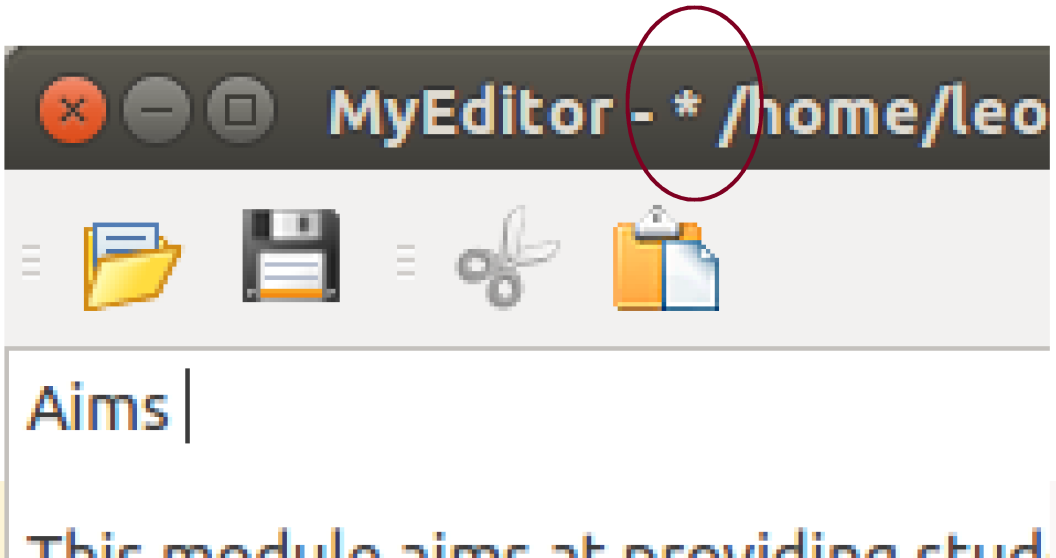
```
str1 == str2
str1 != str2
str1 < str2
str1 > str2
str1.isEmpty()
```

- Formatting

```
QString("Cannot read file %1:\n%2.")
    .arg(fileName).arg(file.errorString())
```

1st argument 2nd argument

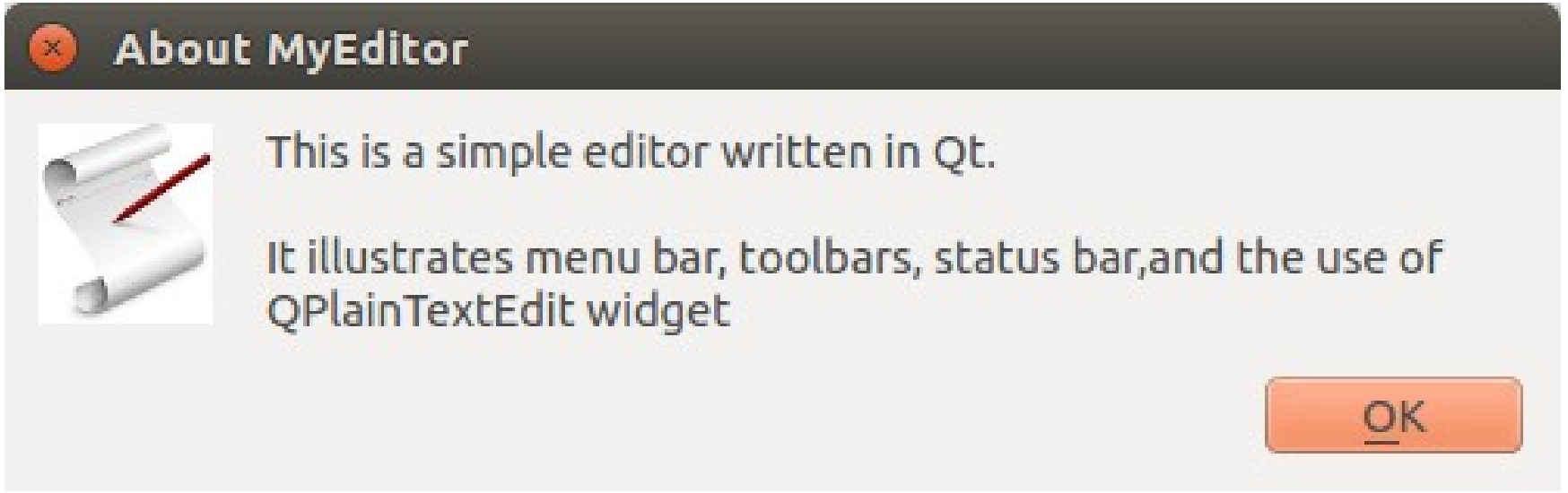
```
void MyEditor::setCurrentFile(const QString &fileName)
{
    currFile = fileName;
    textEdit->document()->setModified(false);
    setWindowModified(false);

    QString showName;
    if (currFile.isEmpty())
        showName = "untitled.txt";
    else
        showName = currFile;

    QString title = "MyEditor - " + "[*]" + showName;
    setWindowTitle(title);
}
```



```
bool MyEditor::save()
{
    if (currFile.isEmpty())
        return false;
    else
        return saveFile(currFile);
}
```

```
bool MyEditor::saveFile(const QString &fileName)
{
    QFile file(fileName);
    if (!file.open(QFile::WriteOnly | QFile::Text))
    {
        QMessageBox::warning(this, tr("MyEditor"),
            tr("Cannot write file %1:\n%2.")
            .arg(fileName).arg(file.errorString()));
        return false;
    }

    QTextStream out(&file);
    QApplication::setOverrideCursor(Qt::WaitCursor);
    out << textEdit->toPlainText();
    QApplication::restoreOverrideCursor();

    setCurrentFile(fileName);
    statusBar()->showMessage(tr("File saved"), 2000);
    return true;
}
```

```
void MyEditor::about()
{
    QMessageBox::about(this, tr("About MyEditor"),
        tr("This is a simple editor written in Qt."
           "It illustrates menu bar, toolbars, status bar,"
           "and the use of QPlainTextEdit widget"));
}
```



```
// main.cpp

#include <QApplication>
#include "MyEditor.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MyEditor editor;
    editor.show();
    return app.exec();  // starts application running
}
```

Memory Management

- Qt implements parent-child mechanism in `QObject`, base class of all Qt objects.
- Deleting parent using `delete` also deletes all children.
  - Makes memory management simpler.
- Deleting a widget also removes it from the screen.
- `MyEditor` inherits `~QWidget()` of `QWidget`.
  - All widgets used are children of `MyEditor`.
  - No need to re-define `~MyEditor()`.

Summary

- `QMainWindow` is a very powerful widget for standard applications.
- Can include or exclude menu, tool bars, docked widgets, etc.

Further Reading

- See complete program in Lab 1.
- Saving main window's settings: [Blan2008] p. 69–70.
- Memory leak: [Blan2008] p. 71–73.
- Splash screen: [Blan2008] p. 74–75.
- Container Classes: [Blan2008] chap. 11.
  - sequential containers: vector, list, linked list
  - associative containers: map, hash table
  - others: string, byte array, variant

References

- J. Blanchette and M. Summerfield, *C++ GUI Programming with Qt 4*, 2nd ed., Prentice Hall, 2008.