ECE 462
Object-Oriented Programming using C++ and Java

Lecture 26

Yung-Hsiang Lu

yunglu@purdue.edu

C++ GUI using Qt

- Use Qt 4 or later. Qt 4.3 is available @msee190pcxx.
- Use your ee462xxx account @msee190xxx.
- Remember to run

  source /home/shay/a/sfwtools/public/settings

  or put it in your ~/.cshrc (if you use csh or tcsh)
- To check whether Qt works, type “qmake -version”.
- Check Qt documentation at

  http://doc.trolltech.com/4.3/index.html
- Many examples at

  /home/shay/a/sfwtools/public/qt4.3.0/examples/

  http://doc.trolltech.com/4.3/examples.html
- Qt4.3 does not work @enad302pcxx.

Install Qt on Your Home Computer

- evaluation version: http://trolltech.com/products/qt/downloads
- open source version:
  - http://trolltech.com/developer/downloads/qt/**windows**
  - http://trolltech.com/developer/downloads/qt/**x11**
  - http://trolltech.com/developer/downloads/qt/**mac**
- Do not try to write GUI from scratch. Learn from examples.
- To create a Qt program:

  > qmake -project

  > qmake

  > make

```
// tutorial 1
#include <QApplication>
#include <QPushButton>
int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  QPushButton hello("Hello world!");
  hello.resize(100, 30);
  hello.show();
  return app.exec();
}
```

```
// t2
#include <QApplication>
#include <QFont>
#include <QPushButton>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QPushButton quitpb("Quit");
    quitpb.resize(75, 30);
    quitpb.setFont(QFont("Times", 18, QFont::Bold));
    QObject::connect(&quitpb, SIGNAL(clicked()),
                &app, SLOT(quit()));
    quitpb.show();
    return app.exec();
}
```

```
// t5
#include <QApplication>
#include <QFont>
#include <QLCDNumber>
#include <QPushButton>
#include <QSlider>
#include <QVBoxLayout>
#include <QWidget>
class MyWidget : public QWidget
{
public:
    MyWidget(QWidget *parent = 0);
};
MyWidget::MyWidget(QWidget *parent)     : QWidget(parent)
{
    QPushButton *quitpb = new QPushButton(tr("Click Me"));
    quitpb->setFont(QFont("Times", 18, QFont::Bold));
```

```
    QLCDNumber *lcd = new QLCDNumber(2);
    lcd->setSegmentStyle(QLCDNumber::Filled);
    QSlider *slider = new QSlider(Qt::Horizontal);
    slider->setRange(0, 99);
    slider->setValue(0);
    connect(quitpb, SIGNAL(clicked()), qApp, SLOT(quit()));
    connect(slider, SIGNAL(valueChanged(int)),
            lcd, SLOT(display(int)));
    QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(quitpb);
    layout->addWidget(lcd);
    layout->addWidget(slider);
    setLayout(layout);
}
```

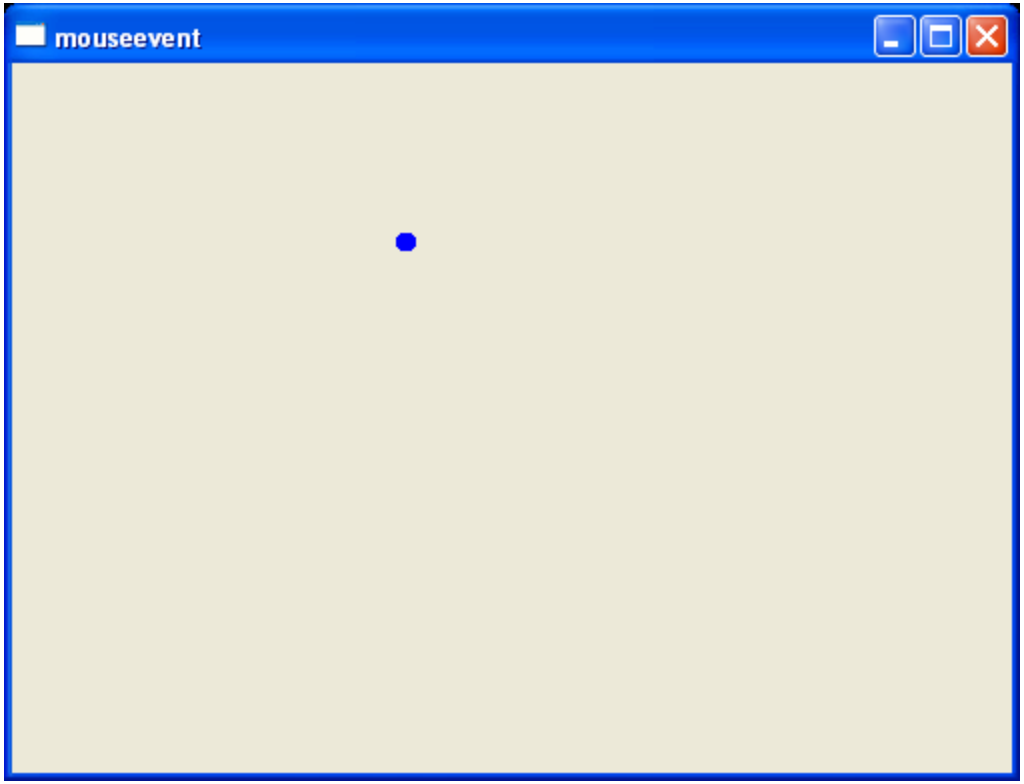
```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MyWidget widget;
    widget.show();
    return app.exec();
}
```

Handle Mouse Event

```
// main.cpp
#include <QApplication>
#include <QFont>
#include <QGridLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QVBoxLayout>
#include "playground.h"
class MyWidget : public QWidget
{
public:
    MyWidget(QWidget *parent = 0);
};
```

```
MyWidget::MyWidget(QWidget *parent) : QWidget(parent)
{
  PlayGround * pg = new PlayGround;
  QGridLayout *gridLayout = new QGridLayout;
  gridLayout->addWidget(pg, 0, 0);
  setLayout(gridLayout);
}

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MyWidget widget;
    widget.setGeometry(100, 100, 500, 355);
    widget.show();
    return app.exec();
}
```


mouseevent

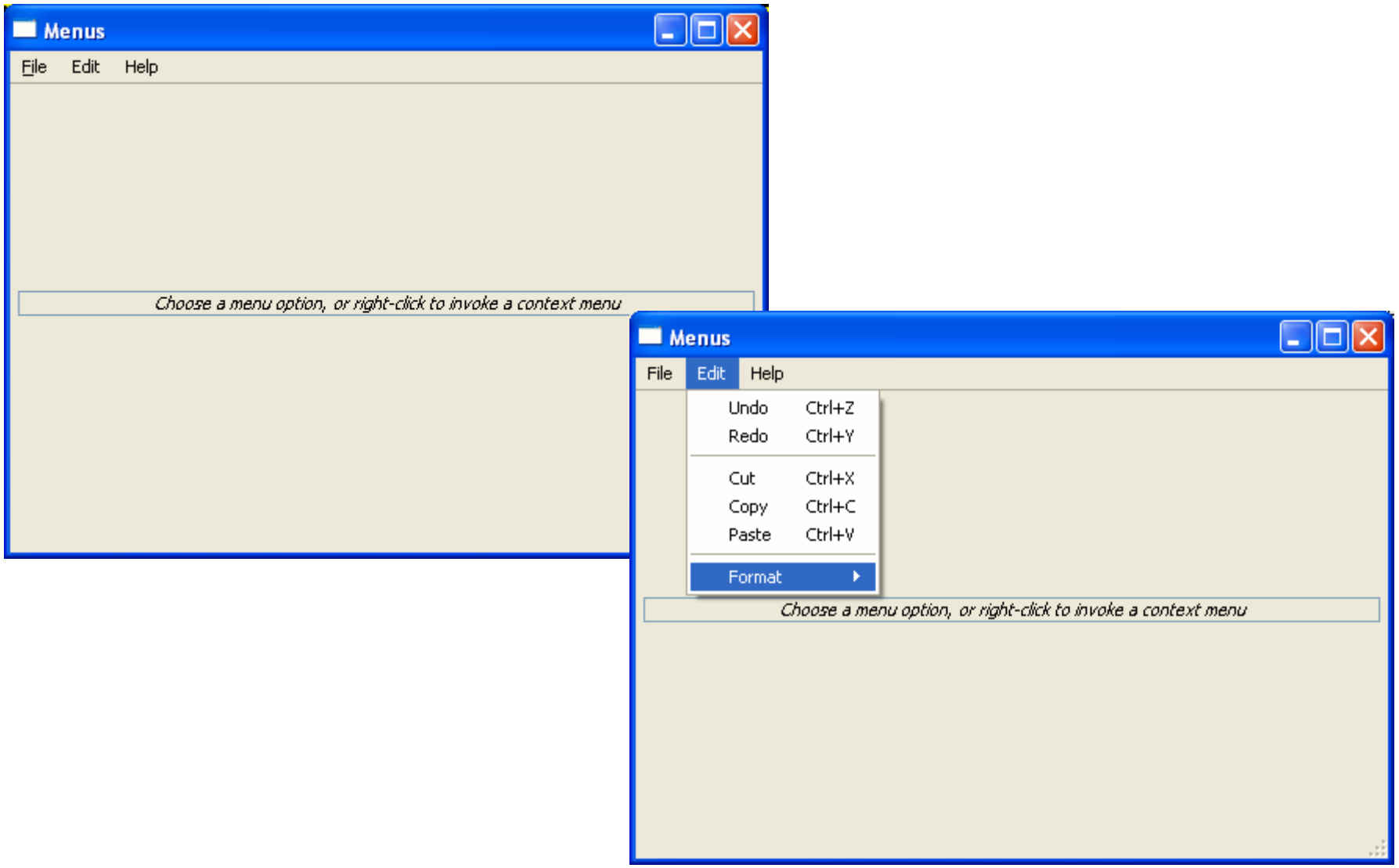
```
// playground.h
#ifndef PLAYGROUND_H
#define PLAYGROUND_H
#include <QWidget>
#include <QPaintEvent>
#include <QMouseEvent>
#include <QPainter>
class PlayGround: public QWidget
{
private:
    int xloc;
    int yloc;
    void paintPG(QPainter &painter);
public:
    PlayGround(QWidget *parent = 0);
```

```
protected:
    void paintEvent(QPaintEvent *event);
    void mouseMoveEvent(QMouseEvent *event);
};
#endif /* PLAYGROUND_H */

// playground.cpp
#include "playground.h"
PlayGround::PlayGround(QWidget *parent): QWidget(parent)
{
  xloc = 100;
  yloc = 100;
}
void PlayGround::paintEvent(QPaintEvent * evt)
{
    QPainter painter(this);
    paintPG(painter);
}
```

```
void PlayGround::paintPG(QPainter &painter)
{
    painter.setPen(Qt::NoPen);
    painter.setBrush(Qt::blue);
    painter.save();
    painter.drawEllipse(xloc, yloc, 10, 10);
    painter.restore();
}

void PlayGround::mouseMoveEvent(QMouseEvent *event)
{
  xloc = event -> x();
  yloc = event -> y();
  update(geometry());
}
```



http://doc.trolltech.com/4.3/mainwindows-menus.html

```
// mainwindow
#include <QApplication>
#include "mainwindow.h"
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MainWindow window;
    window.show();
    return app.exec();
}

// mainwindow.h
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
class QAction;
class QActionGroup;
class QLabel;
class QMenu;
class MainWindow : public
     QMainWindow
{
    Q_OBJECT
public:
    MainWindow();
protected:
    void
     contextMenuEvent(QContextMenu
     Event *event);
private slots:
    void newFile();
    void open();
    void save();
```

```
    void print();
    void undo();
    void redo();
    void cut();
    void copy();
    void paste();
    void bold();
    void italic();
    void leftAlign();
    void rightAlign();
    void justify();
    void center();
    void setLineSpacing();
    void setParagraphSpacing();
    void about();
    void aboutQt();

private:
    void createActions();
    void createMenus();

    QMenu *fileMenu;
    QMenu *editMenu;
    QMenu *formatMenu;
    QMenu *helpMenu;
    QActionGroup *alignmentGroup;
    QAction *newAct;
    QAction *openAct;
    QAction *saveAct;
    QAction *printAct;
    QAction *exitAct;
    QAction *undoAct;
    QAction *redoAct;
    QAction *cutAct;
```

```
    QAction *copyAct;
    QAction *pasteAct;
    QAction *boldAct;
    QAction *italicAct;
    QAction *leftAlignAct;
    QAction *rightAlignAct;
    QAction *justifyAct;
    QAction *centerAct;
    QAction *setLineSpacingAct;
    QAction *setParagraphSpacingAct;
    QAction *aboutAct;
    QAction *aboutQtAct;
    QLabel *infoLabel;
};
#endif
```

```
#include <QtGui>
#include "mainwindow.h"
MainWindow::MainWindow()
{
    QWidget *widget = new QWidget;
    setCentralWidget(widget);

    QWidget *topFiller = new QWidget;
    topFiller->
     setSizePolicy(QSizePolicy::Expan
     ding, SizePolicy::Expanding);

    infoLabel = new
     QLabel(tr("<i>Choose a menu
     option, or right-click to "
                          "invoke a context
     menu</i>"));
```

```
    infoLabel->
     setFrameStyle(QFrame::StyledPa
     nel | QFrame::Sunken);
    infoLabel->
     setAlignment(Qt::AlignCenter);
    QWidget *bottomFiller = new
     QWidget;
    bottomFiller->
     setSizePolicy(QSizePolicy::Expan
     ding, QSizePolicy::Expanding);
    QVBoxLayout *layout = new
     QVBoxLayout;
    layout->setMargin(5);
    layout->addWidget(topFiller);
    layout->addWidget(infoLabel);
    layout->addWidget(bottomFiller);
    widget->setLayout(layout);
    createActions();
    createMenus();
    QString message = tr("A context
     menu is available by right-
     clicking");
    statusBar()->
     showMessage(message);
    setWindowTitle(tr("Menus"));
    setMinimumSize(160, 160);
    resize(480, 320);
}
void
     MainWindow::contextMenuEvent(
     QContextMenuEvent *event) {
    QMenu menu(this);
    menu.addAction(cutAct);
    menu.addAction(copyAct);
    menu.addAction(pasteAct);
    menu.exec(event->globalPos());
}
```

```
void MainWindow::newFile() {
    infoLabel->setText(tr("Invoked
     <b>File|New</b>"));
}
void MainWindow::open() {
    infoLabel->setText(tr("Invoked
     <b>File|Open</b>"));
}
void MainWindow::save() {
    infoLabel->setText(tr("Invoked
     <b>File|Save</b>"));
}
void MainWindow::print() {
    infoLabel->setText(tr("Invoked
     <b>File|Print</b>"));
}
void MainWindow::undo() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Undo</b>"));
}
void MainWindow::redo() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Redo</b>"));
}
void MainWindow::cut() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Cut</b>"));
}
void MainWindow::copy() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Copy</b>"));
}
void MainWindow::paste() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Paste</b>"));
}
```

```
void MainWindow::bold() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Format|Bold</b>"));
}
void MainWindow::italic() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Format|Italic</b>"));
}
void MainWindow::leftAlign() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Format|Left Align</b>"));
}
void MainWindow::rightAlign() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Format|Right Align</b>"));
}
void MainWindow::justify() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Format|Justify</b>"));
}
void MainWindow::center() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Format|Center</b>"));
}
void MainWindow::setLineSpacing() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Format|Set Line
     Spacing</b>"));
}
void
     MainWindow::setParagraphSpacin
     g() {
    infoLabel->setText(tr("Invoked
     <b>Edit|Format|Set Paragraph
     Spacing</b>"));
}
void MainWindow::about() {
    infoLabel->setText(tr("Invoked
     <b>Help|About</b>"));
```

```
QMessageBox::about(this, tr("About
    Menu"),
        tr("The <b>Menu</b>
    example shows how to create "
           "menu-bar menus and
    context menus."));
}
void MainWindow::aboutQt() {
   infoLabel->setText(tr("Invoked
    <b>Help|About Qt</b>"));
}
void MainWindow::createActions() {
   newAct = new QAction(tr("&New"),
    this);
   newAct->setShortcut(tr("Ctrl+N"));
   newAct->setStatusTip(tr("Create a
    new file"));
```

```
connect(newAct,
 SIGNAL(triggered()), this,
 SLOT(newFile()));

openAct = new
 QAction(tr("&Open..."), this);
openAct->setShortcut(tr("Ctrl+O"));
openAct->setStatusTip(tr("Open an
 existing file"));
connect(openAct,
 SIGNAL(triggered()), this,
 SLOT(open()));

saveAct = new
 QAction(tr("&Save"), this);
saveAct->setShortcut(tr("Ctrl+S"));
saveAct->setStatusTip(tr("Save the
 document to disk"));
```

```
 connect(saveAct,
  SIGNAL(triggered()), this,
  SLOT(save()));

printAct = new
  QAction(tr("&Print..."), this);
printAct->setShortcut(tr("Ctrl+P"));
printAct->setStatusTip(tr("Print the
  document"));
connect(printAct,
  SIGNAL(triggered()), this,
  SLOT(print()));

exitAct = new QAction(tr("E&xit"),
  this);
exitAct->setShortcut(tr("Ctrl+Q"));
exitAct->setStatusTip(tr("Exit the
  application"));
```

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

undoAct = new
  QAction(tr("&Undo"), this);
undoAct->setShortcut(tr("Ctrl+Z"));
undoAct->setStatusTip(tr("Undo
  the last operation"));
connect(undoAct,
  SIGNAL(triggered()), this,
  SLOT(undo()));

redoAct = new
  QAction(tr("&Redo"), this);
redoAct->setShortcut(tr("Ctrl+Y"));
redoAct->setStatusTip(tr("Redo the
  last operation"));
connect(redoAct,
  SIGNAL(triggered()), this,
  SLOT(redo()));
```

```
cutAct = new QAction(tr("Cu&t"),
 this);
cutAct->setShortcut(tr("Ctrl+X"));
cutAct->setStatusTip(tr("Cut the
 current selection's contents to the
 ""clipboard"));
connect(cutAct,
 SIGNAL(triggered()), this,
 SLOT(cut()));

copyAct = new
 QAction(tr("&Copy"), this);
copyAct->setShortcut(tr("Ctrl+C"));
copyAct->setStatusTip(tr("Copy the
 current selection's contents to the
 ""clipboard"));
connect(copyAct,
 SIGNAL(triggered()), this,
 SLOT(copy()));
```

```
pasteAct = new
 QAction(tr("&Paste"), this);
pasteAct->setShortcut(tr("Ctrl+V"));
pasteAct->setStatusTip(tr("Paste
 the clipboard's contents into the
 current ""selection"));
connect(pasteAct,
 SIGNAL(triggered()), this,
 SLOT(paste()));

boldAct = new QAction(tr("&Bold"),
 this);
boldAct->setCheckable(true);
boldAct->setShortcut(tr("Ctrl+B"));
boldAct->setStatusTip(tr("Make the
 text bold"));
connect(boldAct,
 SIGNAL(triggered()), this,
 SLOT(bold()));
```

```
QFont boldFont = boldAct->font();
boldFont.setBold(true);
boldAct->setFont(boldFont);

italicAct = new QAction(tr("&Italic"),
 this);
italicAct->setCheckable(true);
italicAct->setShortcut(tr("Ctrl+I"));
italicAct->setStatusTip(tr("Make the
 text italic"));
connect(italicAct,
 SIGNAL(triggered()), this,
 SLOT(italic()));

QFont italicFont = italicAct->font();
italicFont.setItalic(true);
italicAct->setFont(italicFont);

setLineSpacingAct = new
 QAction(tr("Set &Line Spacing..."),
 this);
setLineSpacingAct->
 setStatusTip(tr("Change the gap
 between the lines of a
 ""paragraph"));
connect(setLineSpacingAct,
 SIGNAL(triggered()), this,
 SLOT(setLineSpacing()));
setParagraphSpacingAct = new
 QAction(tr("Set &Paragraph
 Spacing..."), this);
setLineSpacingAct->
 setStatusTip(tr("Change the gap
 between paragraphs"));
connect(setParagraphSpacingAct,
 SIGNAL(triggered()),   this,
 SLOT(setParagraphSpacing()));
```

```
aboutAct = new
 QAction(tr("&About"), this);
aboutAct->setStatusTip(tr("Show
 the application's About box"));
connect(aboutAct,
 SIGNAL(triggered()), this,
 SLOT(about()));

aboutQtAct = new
 QAction(tr("About &Qt"), this);
aboutQtAct->setStatusTip(tr("Show
 the Qt library's About box"));
connect(aboutQtAct,
 SIGNAL(triggered()), qApp,
 SLOT(aboutQt()));
connect(aboutQtAct,
 SIGNAL(triggered()), this,
 SLOT(aboutQt()));
```

```
leftAlignAct = new
 QAction(tr("&Left Align"), this);
leftAlignAct->setCheckable(true);
leftAlignAct->
 setShortcut(tr("Ctrl+L"));
leftAlignAct->setStatusTip(tr("Left
 align the selected text"));
connect(leftAlignAct,
 SIGNAL(triggered()), this,
 SLOT(leftAlign()));

rightAlignAct = new
 QAction(tr("&Right Align"), this);
rightAlignAct->setCheckable(true);
rightAlignAct->
 setShortcut(tr("Ctrl+R"));
rightAlignAct->
 setStatusTip(tr("Right align the
 selected text"));
```

```
    connect(rightAlignAct,
     SIGNAL(triggered()), this,
     SLOT(rightAlign()));
    justifyAct = new
     QAction(tr("&Justify"), this);
    justifyAct->setCheckable(true);
    justifyAct->setShortcut(tr("Ctrl+J"));
    justifyAct->setStatusTip(tr("Justify
     the selected text"));
    connect(justifyAct,
     SIGNAL(triggered()), this,
     SLOT(justify()));
    centerAct = new
     QAction(tr("&Center"), this);
    centerAct->setCheckable(true);
    centerAct->
     setShortcut(tr("Ctrl+E"));
    centerAct->setStatusTip(tr("Center
     the selected text"));
    connect(centerAct,
     SIGNAL(triggered()), this,
     SLOT(center()));
    alignmentGroup = new
     QActionGroup(this);
    alignmentGroup-
     >addAction(leftAlignAct);
    alignmentGroup-
     >addAction(rightAlignAct);
    alignmentGroup-
     >addAction(justifyAct);
    alignmentGroup-
     >addAction(centerAct);
    leftAlignAct->setChecked(true);
}
void MainWindow::createMenus() {
    fileMenu = menuBar()->
     addMenu(tr("&File"));
```

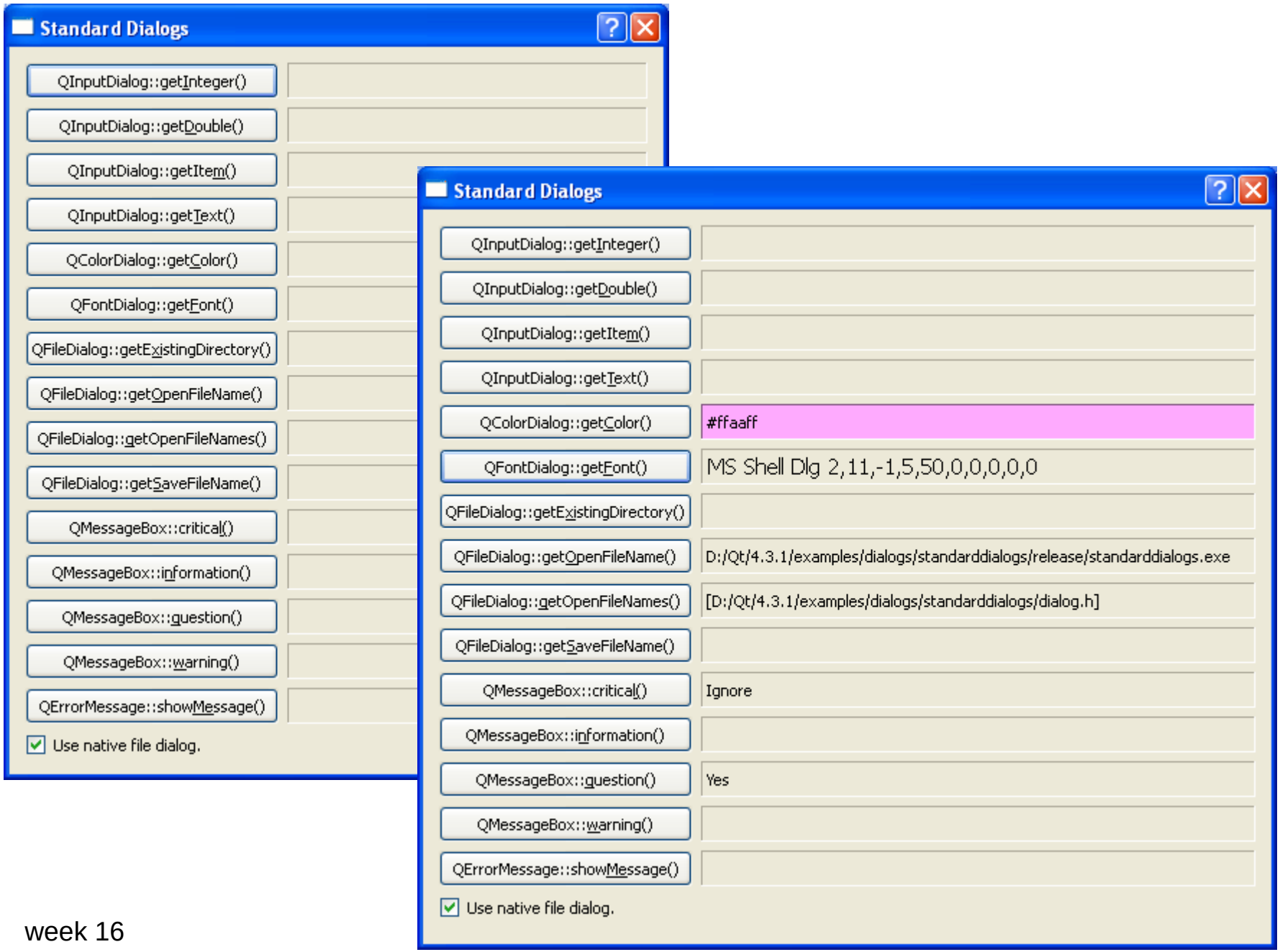
```
fileMenu->addAction(newAct);
fileMenu->addAction(openAct);
fileMenu->addAction(saveAct);
fileMenu->addAction(printAct);
fileMenu->addSeparator();
fileMenu->addAction(exitAct);

editMenu = menuBar()-
  >addMenu(tr("&Edit"));
editMenu->addAction(undoAct);
editMenu->addAction(redoAct);
editMenu->addSeparator();
editMenu->addAction(cutAct);
editMenu->addAction(copyAct);
editMenu->addAction(pasteAct);
editMenu->addSeparator();
```

```
helpMenu = menuBar()-
  >addMenu(tr("&Help"));
helpMenu->addAction(aboutAct);
helpMenu->addAction(aboutQtAct);

formatMenu = editMenu-
  >addMenu(tr("&Format"));
formatMenu->addAction(boldAct);
formatMenu->addAction(italicAct);
formatMenu->addSeparator()-
  >setText(tr("Alignment"));
formatMenu-
  >addAction(leftAlignAct);
formatMenu-
  >addAction(rightAlignAct);
formatMenu->addAction(justifyAct);
formatMenu-
  >addAction(centerAct);
formatMenu->addSeparator();
```

```
    formatMenu-> addAction(setLineSpacingAct);
    formatMenu->addAction(setParagraphSpacingAct);
}
```


Standard Dialogs
QInputDialog::getInteger()
QInputDialog::getDouble()
QInputDialog::getItem()
QInputDialog::getText()
QColorDialog::getColor()
QFontDialog::getFont()
QFileDialog::getExistingDirectory()
QFileDialog::getOpenFileName()
QFileDialog::getOpenFileNames()
QFileDialog::getSaveFileName()
QMessageBox::critical()
QMessageBox::information()
QMessageBox::question()
QMessageBox::warning()
QErrorMessage::showMessage()
Use native file dialog.
Standard Dialogs
QInputDialog::getInteger()
QInputDialog::getDouble()
QInputDialog::getItem()
QInputDialog::getText()
QColorDialog::getColor()
#ffaaff
QFontDialog::getFont()
MS Shell Dlg 2,11,-1,5,50,0,0,0,0,0
QFileDialog::getExistingDirectory()
QFileDialog::getOpenFileName()
D:/Qt/4.3.1/examples/dialogs/standarddialogs/release/standarddialogs.exe
QFileDialog::getOpenFileNames()
[D:/Qt/4.3.1/examples/dialogs/standarddialogs/dialog.h]
QFileDialog::getSaveFileName()
QMessageBox::critical()
Ignore
QMessageBox::information()
QMessageBox::question()
Yes
QMessageBox::warning()
QErrorMessage::showMessage()
Use native file dialog.

week 16

```
#include <QApplication>
#include <QTranslator>
#include <QLocale>
#include <QLibraryInfo>
#include "dialog.h"
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QString translatorFileName = QLatin1String("qt_");
    translatorFileName += QLocale::system().name();
    QTranslator *translator = new QTranslator(&app);
    if (translator->load(translatorFileName,
     QLibraryInfo::location(QLibraryInfo::TranslationsPath)))
        app.installTranslator(translator);
    Dialog dialog;
    return dialog.exec();
}
```

```
#ifndef DIALOG_H
#define DIALOG_H
#include <QDialog>
class QCheckBox;
class QLabel;
class QErrorMessage;
class Dialog : public QDialog
{
    Q_OBJECT
public:
    Dialog(QWidget *parent = 0);
private slots:
    void setInteger();
    void setDouble();
    void setItem();
    void setText();
    void setColor();
    void setFont();
    void setExistingDirectory();
    void setOpenFileName();
    void setOpenFileNames();
    void setSaveFileName();
    void criticalMessage();
    void informationMessage();
    void questionMessage();
    void warningMessage();
    void errorMessage();
private:
    QCheckBox *native;
    QLabel *integerLabel;
    QLabel *doubleLabel;
    QLabel *itemLabel;
    QLabel *textLabel;
    QLabel *colorLabel;
```

```
    QLabel *fontLabel;
    QLabel *directoryLabel;
    QLabel *openFileNameLabel;
    QLabel *openFileNamesLabel;
    QLabel *saveFileNameLabel;
    QLabel *criticalLabel;
    QLabel *informationLabel;
    QLabel *questionLabel;
    QLabel *warningLabel;
    QLabel *errorLabel;
    QErrorMessage
     *errorMessageDialog;

    QString openFilesPath;
};
#endif
```

```
#include <QtGui>
#include "dialog.h"
#define MESSAGE \
    Dialog::tr("<p>Message boxes have a caption, a text, " \
               "and any number of buttons, each with standard or custom texts." \
               "<p>Click a button to close the message box. Pressing the Esc button
     " \
               "will activate the detected escape button (if any).")
Dialog::Dialog(QWidget *parent)
    : QDialog(parent)
{
    errorMessageDialog = new QErrorMessage(this);
    int frameStyle = QFrame::Sunken | QFrame::Panel;
    integerLabel = new QLabel;
    integerLabel->setFrameStyle(frameStyle);
    QPushButton *integerButton =
            new QPushButton(tr("QInputDialog::get&Integer()"));
```

```
doubleLabel = new QLabel;
doubleLabel->setFrameStyle(frameStyle);
QPushButton *doubleButton =
        new QPushButton(tr("QInputDialog::get&Double()"));
itemLabel = new QLabel;
itemLabel->setFrameStyle(frameStyle);
QPushButton *itemButton = new
  QPushButton(tr("QInputDialog::getIte&m()"));
textLabel = new QLabel;
textLabel->setFrameStyle(frameStyle);
QPushButton *textButton = new
  QPushButton(tr("QInputDialog::get&Text()"));
colorLabel = new QLabel;
colorLabel->setFrameStyle(frameStyle);
QPushButton *colorButton = new
  QPushButton(tr("QColorDialog::get&Color()"));
fontLabel = new QLabel;
```

```
    fontLabel->setFrameStyle(frameStyle);
    QPushButton *fontButton = new
     QPushButton(tr("QFontDialog::get&Font()"));
    directoryLabel = new QLabel;
    directoryLabel->setFrameStyle(frameStyle);
    QPushButton *directoryButton =
            new QPushButton(tr("QFileDialog::getE&xistingDirectory()"));
    openFileNameLabel = new QLabel;
    openFileNameLabel->setFrameStyle(frameStyle);
    QPushButton *openFileNameButton =
            new QPushButton(tr("QFileDialog::get&OpenFileName()"));
    openFileNamesLabel = new QLabel;
    openFileNamesLabel->setFrameStyle(frameStyle);
    QPushButton *openFileNamesButton =
            new QPushButton(tr("QFileDialog::&getOpenFileNames()"));
    saveFileNameLabel = new QLabel;
    saveFileNameLabel->setFrameStyle(frameStyle);
    QPushButton *saveFileNameButton =
    new QPushButton(tr("QFileDialog::get&SaveFileName()"));
```

```
criticalLabel = new QLabel;
criticalLabel->setFrameStyle(frameStyle);
QPushButton *criticalButton =
        new QPushButton(tr("QMessageBox::critica&l()"));
informationLabel = new QLabel;
informationLabel->setFrameStyle(frameStyle);
QPushButton *informationButton =
        new QPushButton(tr("QMessageBox::i&nformation()"));
questionLabel = new QLabel;
questionLabel->setFrameStyle(frameStyle);
QPushButton *questionButton =
        new QPushButton(tr("QMessageBox::&question()"));
warningLabel = new QLabel;
warningLabel->setFrameStyle(frameStyle);
QPushButton *warningButton = new
  QPushButton(tr("QMessageBox::&warning()"));
errorLabel = new QLabel;
errorLabel->setFrameStyle(frameStyle);
```

```
    QPushButton *errorButton =
            new QPushButton(tr("QErrorMessage::show&M&essage()"));
    connect(integerButton, SIGNAL(clicked()), this, SLOT(setInteger()));
    connect(doubleButton, SIGNAL(clicked()), this, SLOT(setDouble()));
    connect(itemButton, SIGNAL(clicked()), this, SLOT(setItem()));
    connect(textButton, SIGNAL(clicked()), this, SLOT(setText()));
    connect(colorButton, SIGNAL(clicked()), this, SLOT(setColor()));
    connect(fontButton, SIGNAL(clicked()), this, SLOT(setFont()));
    connect(directoryButton, SIGNAL(clicked()),
            this, SLOT(setExistingDirectory()));
    connect(openFileNameButton, SIGNAL(clicked()),
            this, SLOT(setOpenFileName()));
    connect(openFileNamesButton, SIGNAL(clicked()),
            this, SLOT(setOpenFileNames()));
    connect(saveFileNameButton, SIGNAL(clicked()),
            this, SLOT(setSaveFileName()));
    connect(criticalButton, SIGNAL(clicked()), this, SLOT(criticalMessage()));
    connect(informationButton, SIGNAL(clicked()),
            this, SLOT(informationMessage()));
```

```
    connect(questionButton, SIGNAL(clicked()), this, SLOT(questionMessage()));
    connect(warningButton, SIGNAL(clicked()), this, SLOT(warningMessage()));
    connect(errorButton, SIGNAL(clicked()), this, SLOT(errorMessage()));
    native = new QCheckBox(this);
    native->setText("Use native file dialog.");
    native->setChecked(true);
#ifndef Q_WS_WIN
#ifndef Q_OS_MAC
    native->hide();
#endif
#endif
    QGridLayout *layout = new QGridLayout;
    layout->setColumnStretch(1, 1);
    layout->setColumnMinimumWidth(1, 250);
    layout->addWidget(integerButton, 0, 0);
    layout->addWidget(integerLabel, 0, 1);
    layout->addWidget(doubleButton, 1, 0);
    layout->addWidget(doubleLabel, 1, 1);
```

```
    layout->addWidget(itemButton, 2, 0);
    layout->addWidget(itemLabel, 2, 1);
    layout->addWidget(textButton, 3, 0);
    layout->addWidget(textLabel, 3, 1);
    layout->addWidget(colorButton, 4, 0);
    layout->addWidget(colorLabel, 4, 1);
    layout->addWidget(fontButton, 5, 0);
    layout->addWidget(fontLabel, 5, 1);
    layout->addWidget(directoryButton, 6, 0);
    layout->addWidget(directoryLabel, 6, 1);
    layout->addWidget(openFileNameButton, 7, 0);
    layout->addWidget(openFileNameLabel, 7, 1);
    layout->addWidget(openFileNamesButton, 8, 0);
    layout->addWidget(openFileNamesLabel, 8, 1);
    layout->addWidget(saveFileNameButton, 9, 0);
    layout->addWidget(saveFileNameLabel, 9, 1);
    layout->addWidget(criticalButton, 10, 0);
    layout->addWidget(criticalLabel, 10, 1);
```

```
    layout->addWidget(informationButton, 11, 0);
    layout->addWidget(informationLabel, 11, 1);
    layout->addWidget(questionButton, 12, 0);
    layout->addWidget(questionLabel, 12, 1);
    layout->addWidget(warningButton, 13, 0);
    layout->addWidget(warningLabel, 13, 1);
    layout->addWidget(errorButton, 14, 0);
    layout->addWidget(errorLabel, 14, 1);
    layout->addWidget(native, 15, 0);
    setLayout(layout);
    setWindowTitle(tr("Standard Dialogs"));
}
void Dialog::setInteger()
{
    bool ok;
    int i = QInputDialog::getInteger(this, tr("QInputDialog::getInteger()"),
                                     tr("Percentage:"), 25, 0, 100, 1, &ok);
```

```
if (ok)
        integerLabel->setText(tr("%1%").arg(i));
}
void Dialog::setDouble()
{
    bool ok;
    double d = QInputDialog::getDouble(this, tr("QInputDialog::getDouble()"),
                                       tr("Amount:"), 37.56, -10000, 10000, 2, &ok);
    if (ok)
        doubleLabel->setText(QString("$%1").arg(d));
}
void Dialog::setItem()
{
    QStringList items;
    items << tr("Spring") << tr("Summer") << tr("Fall") << tr("Winter");
    bool ok;
    QString item = QInputDialog::getItem(this, tr("QInputDialog::getItem()"),
                                         tr("Season:"), items, 0, false, &ok);
```

```
if (ok && !item.isEmpty())
        itemLabel->setText(item);
}
void Dialog::setText()
{
    bool ok;
    QString text = QInputDialog::getText(this, tr("QInputDialog::getText()"),
                                         tr("User name:"), QLineEdit::Normal,
                                         QDir::home().dirName(), &ok);
    if (ok && !text.isEmpty())
        textLabel->setText(text);
}
void Dialog::setColor()
{
    QColor color = QColorDialog::getColor(Qt::green, this);
    if (color.isValid()) {
        colorLabel->setText(color.name());
        colorLabel->setPalette(QPalette(color));
```

```
colorLabel->setAutoFillBackground(true);
    }
}
void Dialog::setFont()
{
    bool ok;
    QFont font = QFontDialog::getFont(&ok, QFont(fontLabel->text()), this);
    if (ok) {
        fontLabel->setText(font.key());
        fontLabel->setFont(font);
    }
}
void Dialog::setExistingDirectory()
{
    QFileDialog::Options options = QFileDialog::DontResolveSymlinks |
     QFileDialog::ShowDirsOnly;
    if (!native->isChecked())
        options |= QFileDialog::DontUseNativeDialog;
```

```
QString directory = QFileDialog::getExistingDirectory(this,
                                tr("QFileDialog::getExistingDirectory()"),
                                directoryLabel->text(),
                                options);
    if (!directory.isEmpty())
        directoryLabel->setText(directory);
}
void Dialog::setOpenFileName()
{
    QFileDialog::Options options;
    if (!native->isChecked())
        options |= QFileDialog::DontUseNativeDialog;
    QString selectedFilter;
    QString fileName = QFileDialog::getOpenFileName(this,
                                tr("QFileDialog::getOpenFileName()"),
                                openFileNameLabel->text(),
                                tr("All Files (*);;Text Files (*.txt)"),
                                &selectedFilter, options);
```

```
if (!fileName.isEmpty())
        openFileNameLabel->setText(fileName);
}
void Dialog::setOpenFileNames()
{
    QFileDialog::Options options;
    if (!native->isChecked())
        options |= QFileDialog::DontUseNativeDialog;
    QString selectedFilter;
    QStringList files = QFileDialog::getOpenFileNames(
                            this, tr("QFileDialog::getOpenFileNames()"),
                            openFilesPath,  tr("All Files (*);;Text Files (*.txt)"),
                            &selectedFilter, options);
    if (files.count()) {
        openFilesPath = files[0];
        openFileNamesLabel->setText(QString("[%1]").arg(files.join(", ")));
    }
}
```

```
void Dialog::setSaveFileName()
{
    QFileDialog::Options options;
    if (!native->isChecked())
        options |= QFileDialog::DontUseNativeDialog;
    QString selectedFilter;
    QString fileName = QFileDialog::getSaveFileName(this,
                                tr("QFileDialog::getSaveFileName()"),
                                saveFileNameLabel->text(),
                                tr("All Files (*);;Text Files (*.txt)"),
                                &selectedFilter,
                                options);
    if (!fileName.isEmpty())
        saveFileNameLabel->setText(fileName);
}
void Dialog::criticalMessage()
{
```

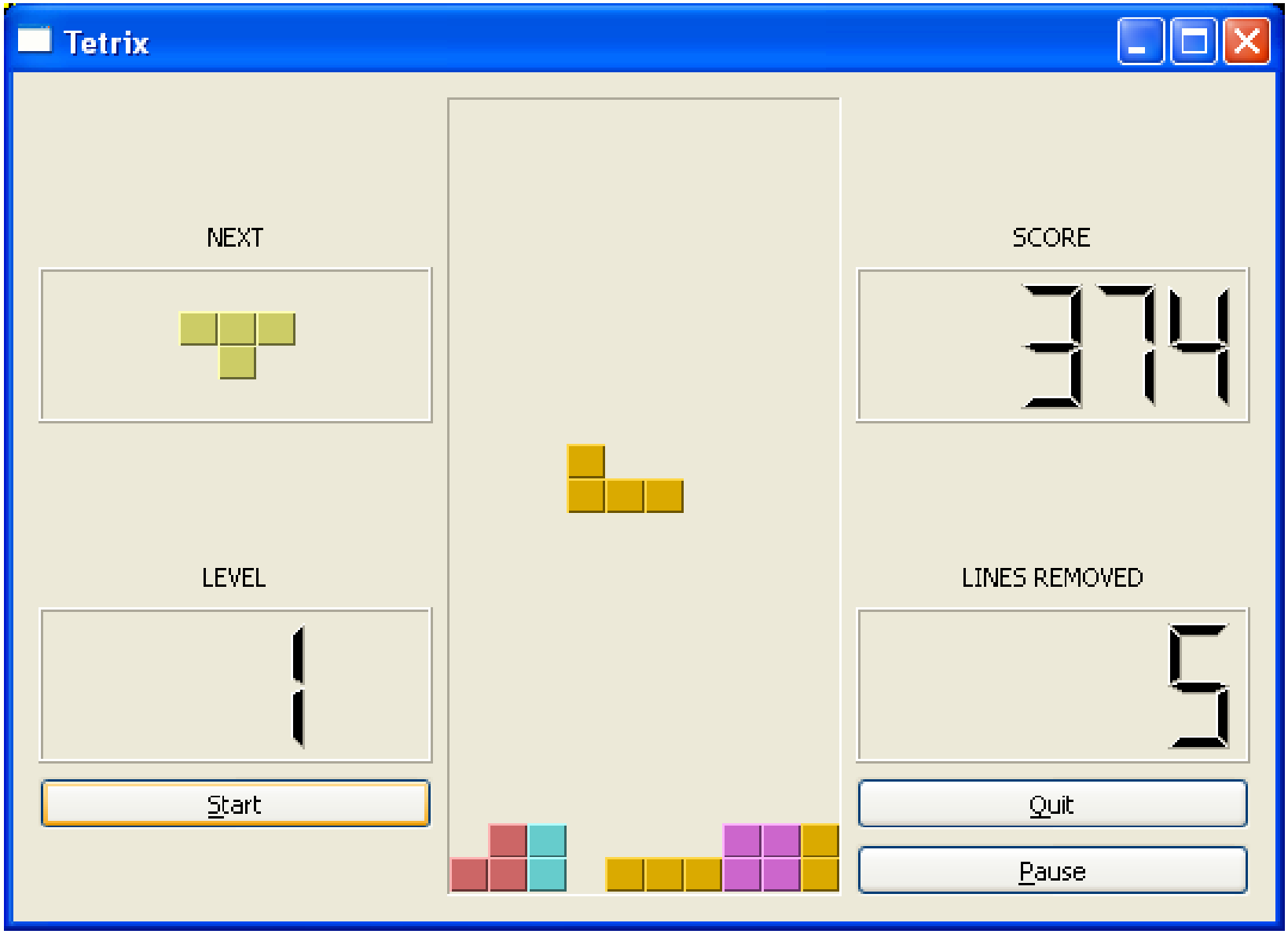
```
  QMessageBox::StandardButton reply;
    reply = QMessageBox::critical(this, tr("QMessageBox::critical()"),
                                  MESSAGE,
                                  QMessageBox::Abort | QMessageBox::Retry |
     QMessageBox::Ignore);
    if (reply == QMessageBox::Abort)
        criticalLabel->setText(tr("Abort"));
    else if (reply == QMessageBox::Retry)
        criticalLabel->setText(tr("Retry"));
    else
        criticalLabel->setText(tr("Ignore"));
}
void Dialog::informationMessage()
{
    QMessageBox::StandardButton reply;
    reply = QMessageBox::information(this, tr("QMessageBox::information()"),
     MESSAGE);
```

```
if (reply == QMessageBox::Ok)
        informationLabel->setText(tr("OK"));
    else
        informationLabel->setText(tr("Escape"));
}
void Dialog::questionMessage()
{
    QMessageBox::StandardButton reply;
    reply = QMessageBox::question(this, tr("QMessageBox::question()"),
                                  MESSAGE, QMessageBox::Yes | QMessageBox::No |
     QMessageBox::Cancel);
    if (reply == QMessageBox::Yes)
        questionLabel->setText(tr("Yes"));
    else if (reply == QMessageBox::No)
        questionLabel->setText(tr("No"));
    else
        questionLabel->setText(tr("Cancel"));
```

```
}
void Dialog::warningMessage()
{
    QMessageBox msgBox(QMessageBox::Warning,
     tr("QMessageBox::warning()"),
                    MESSAGE, 0, this);
    msgBox.addButton(tr("Save &Again"), QMessageBox::AcceptRole);
    msgBox.addButton(tr("&Continue"), QMessageBox::RejectRole);
    if (msgBox.exec() == QMessageBox::AcceptRole)
        warningLabel->setText(tr("Save Again"));
    else
        warningLabel->setText(tr("Continue"));

}
```

```
void Dialog::errorMessage()
{
    errorMessageDialog->showMessage(
            tr("This dialog shows and remembers error messages. "
               "If the checkbox is checked (as it is by default), "
               "the shown message will be shown again, "
               "but if the user unchecks the box the message "
               "will not appear again if QErrorMessage::showMessage() "
               "is called with the same message."));
    errorLabel->setText(tr("If the box is unchecked, the message "
                           "won't appear again."));
}
```


Tetrix
NEXT
SCORE
374
LEVEL
1
LINES REMOVED
5
Start
Quit
Pause

Suggestions about Finding a Job

- List "keywords" in you resume, such as C++, Java, Networking, Thread ...
- Give some technical depth about your knowledge:
  - Course Taken: C++ and Java
  - C++ and Java:

1. Develop an interactive game in Java using Netbeans. The graphical user interface (GUI) includes two buttons, a slider, and is controlled by mouse motion.
2. Create a two-player game using networking ...
3. Use multi-thread ... and obtain 7.2 times speedup on a QuadCore x 2 machine ...

Prepare and Be Active

- Your interview does not have to be passive, responding to questions. You can be active by providing the information about yourself. Preparation is crucial.
- Sometimes, an interviewer may ask you open questions, such as
  - Tell me what you know about Java.
  - Describe your experience writing C++ code.
  - What types of programs have you written using multiple threads?

Bring Your "Cheat Sheets"

- Buy a binder with sheet protectors

- Bring several copies of your resume. Put them at the front of the binder.
- Print GUIs (in color), UML diagrams, key concepts of your design / implementation.
- Be ready to fill in the silence and explain your accomplishment.
- Do not bring 40-page of source code.