Advanced QUiLoader Mastery Exam

Qt Developer

June 14, 2025

Exam Overview

Implement a dynamic UI management system using `QUiLoader` that:

- Loads a main window with plugin management capabilities
- Handles cross-widget communication
- Implements memory-safe dynamic loading

1 Problem Statement

Develop an application that:

1.1 Core Requirements

1. Loads `main_window.ui` containing:
    - A `QStackedWidget` (container area)
    - A `QComboBox` (plugin selector)
    - A "Reload" button
2. Scans a `plugins/` directory for UI files
3. Implements the following features:

2 Exam Questions

2.1 Question 1: Dynamic Loading (5pts)

Implement:

```
// 1. Load main_window.ui using QUiLoader
QUiLoader loader;
QWidget *mainWindow = /* ... */;

// 2. Populate QComboBox with .ui files from plugins/
```

```cpp
QDir pluginsDir("plugins");
QStringList uiFiles = /* ... */;

// 3. Load selected UI into QStackedWidget
void loadPlugin(const QString &filePath) {
    // Your implementation
}
```

2.2 Question 2: Event Handling (6pts)

- Connect `QComboBox` to:
  1. Auto-load selected UI
  2. Update window title with filename
- Make "Reload" button:
  1. Preserve form data during reload
  2. Use `QVariantMap` to save/restore state

2.3 Question 3: Cross-Widget Communication (4pts)

If loaded UI contains:

```cpp
QLineEdit *searchInput = findChild<QLineEdit*>("searchInput");
QListView *listView = mainWindow->findChild<QListView*>("listView")
    ;
```

Implement real-time filtering:

1. Connect `textChanged()` to filter `listView`
2. Use `QSortFilterProxyModel` for advanced filtering

2.4 Question 4: Memory Management (3pts)

- Use `QPointer` for all dynamic widgets
- Implement LRU cache for 3 most recent UIs
- Properly cleanup unused UIs with `deleteLater()`

2.5 Question 5: Advanced Extension (2pts Bonus)

For UIs containing:

```cpp
QWidget *settingsPanel = findChild<QWidget*>("settingsPanel");
```

Add:

- Animated overlay with `QPropertyAnimation`
- Settings toggle button in main window

Grading Rubric

Criteria	Points
Working dynamic loading system	5
Proper event handling	6
Cross-widget communication	4
Memory safety	3
Advanced features (bonus)	2

Key QUiLoader Methods

Implement these techniques:

- `loader.registerCustomWidget()` for custom components
- `loader.errorString()` for debugging
- `loader.setWorkingDirectory()` for relative paths
- Loading from `QBuffer` for in-memory UI files

Tips

- Test with these sample UIs:
  - `list_view.ui` (simple QListView)
  - `form.ui` (input fields)
  - `dashboard.ui` (complex layout)
- Debug with:

```
    qDebug() << "Active widgets:" << mainWindow->findChildren<
    QWidget*>();

```