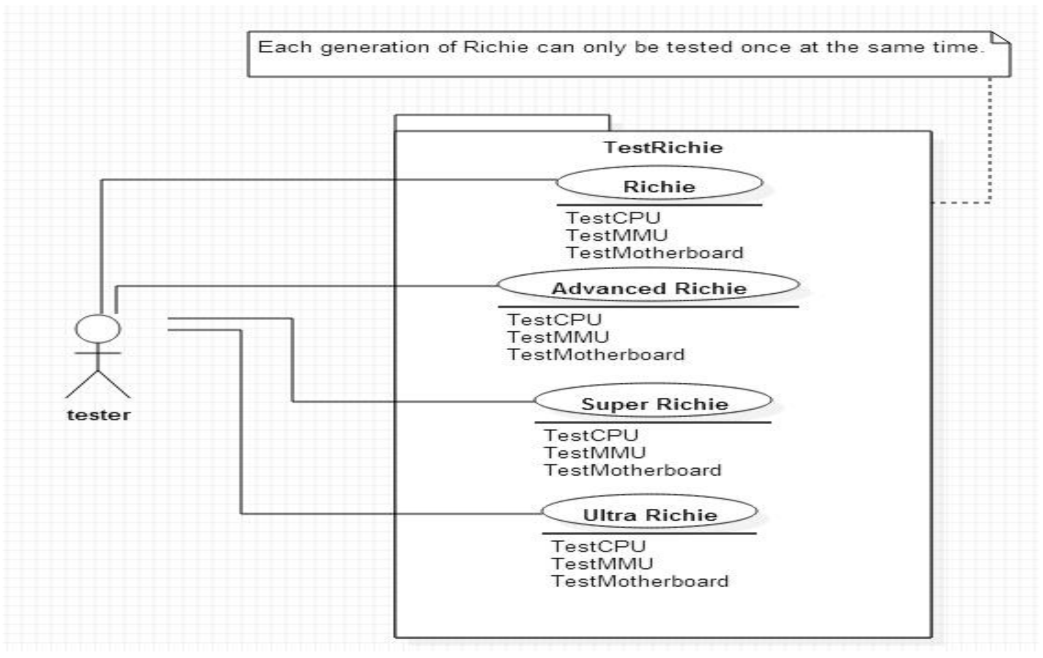
Class Project 1 Report

Yu Chen

1. Problem Statement

Writing a program to test each one of the core components (namely, CPU, MMU and Motherboard) of each one of the generations of Richie (a series of computers, namely, Richie, Advanced Richie, Super Richie and Ultra Richie). Each generation of Richie can only be tested once at the same time.

2. Requirement Specification

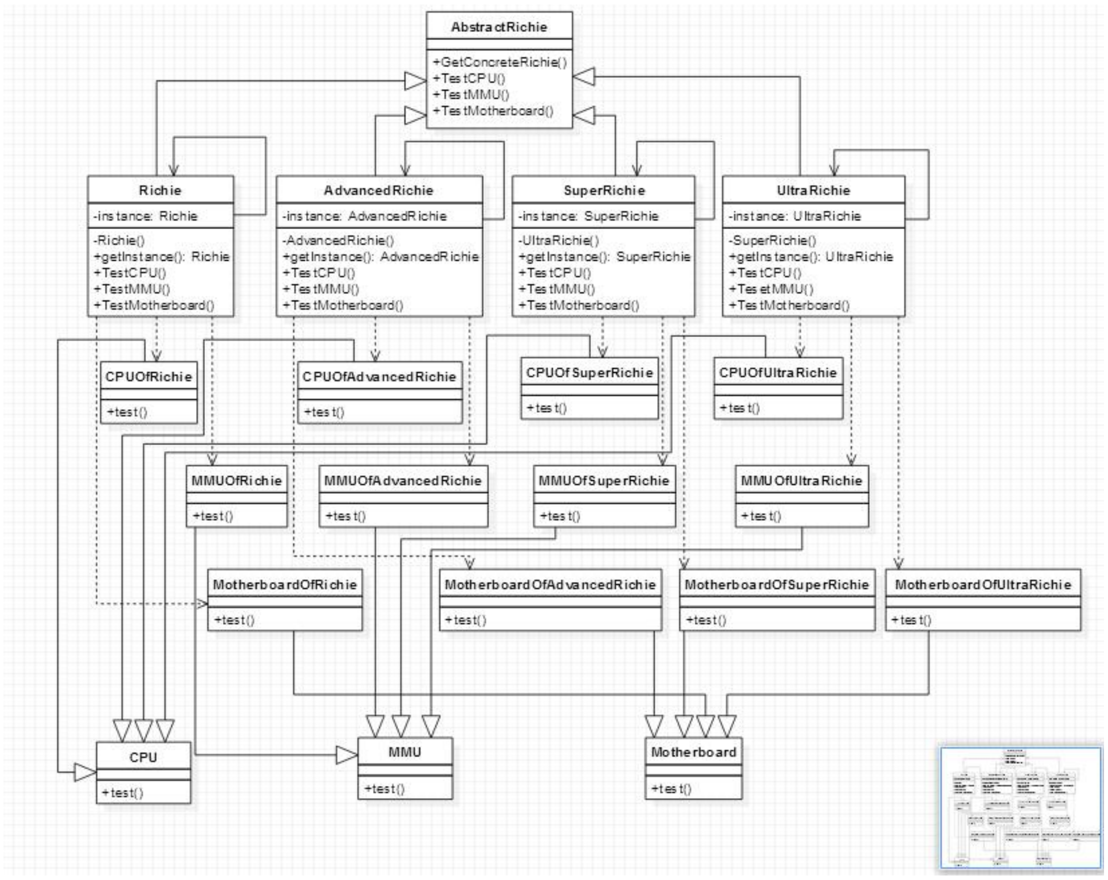


3. Testing Strategy

Categories of what to test:

- ✓ Test Richie: CPU of Richie, MMU of Richie, Motherboard of Richie.
- ✓ Test Advanced Richie: CPU of Advanced Richie, MMU of Advanced Richie and Motherboard of Advanced Richie.
- ✓ Test Super Richie: CPU of Super Richie, MMU of Super Richie and Motherboard of Super Richie.
- ✓ Test Ultra Richie: CPU of Ultra Richie, MMU of Ultra Richie and Motherboard of Ultra Richie.
- ✓ Singleton: Richie, Advanced Richie, Super Richie, Ultra Richie.

4. Design



5. Algorithm

Using the abstract factory pattern to organize the creation of objects that correspond to core computer components and the singleton pattern to ensure that the instance of each concrete factory is unique.

6. Test Plan

Test Richie: instantiate a concrete factory – Richie, call TestCPU() method, TestMMU() method and TestMotherboard() method.

Test Advanced Richie: instantiate a concrete factory – AdvancedRichie, call TestCPU() method, TestMMU() method and TestMotherboard() method.

Test Super Richie: instantiate a concrete factory – SuperRichie, call TestCPU() method, TestMMU() method and TestMotherboard() method.

Test Ultra Richie: instantiate a concrete factory – UltraRichie, call TestCPU() method, TestMMU() method and TestMotherboard() method.

Test Singleton: Richie, Advanced Richie, Super Richie and Ultra Richie.

7. Code

See source code file.

8. Test plan with results

Test Richie: good

Test Advanced Richie: good

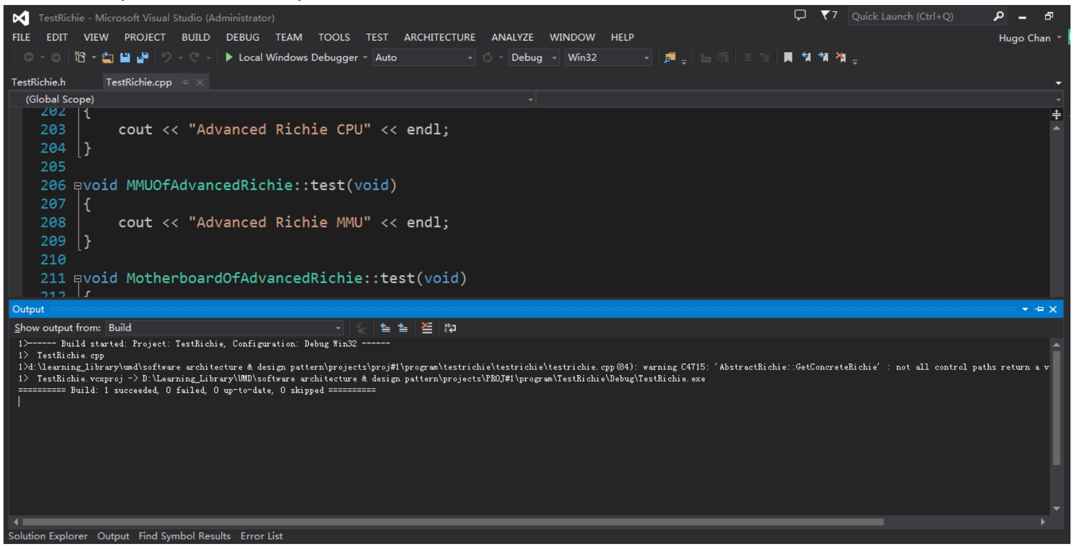
Test Super Richie: good

Test Ultra Richie: good

Test Singleton: good

9. Output screen shots

- Compile/build output



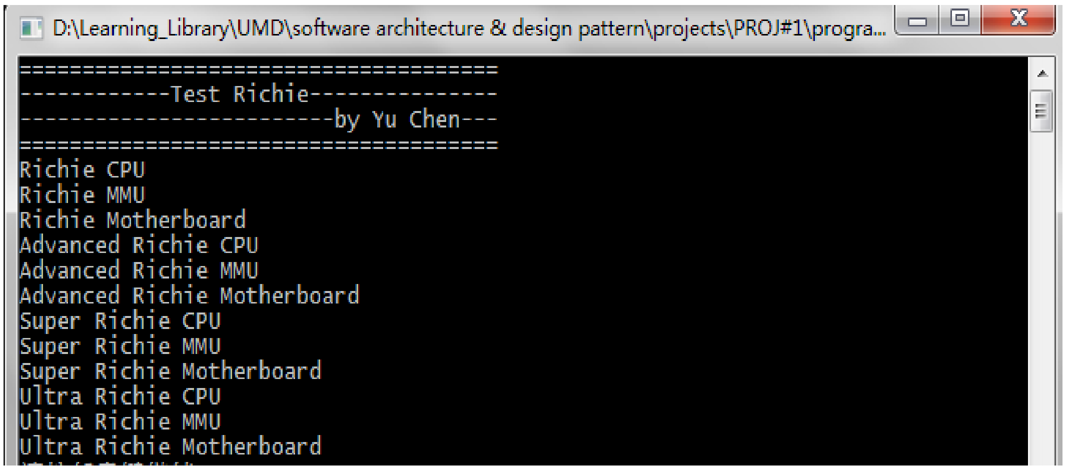
```
    {
        cout << "Advanced Richie CPU" << endl;
    }

void MMUOfAdvancedRichie::test(void)
    {
        cout << "Advanced Richie MMU" << endl;
    }

void MotherboardOfAdvancedRichie::test(void)
```

```
1>------ Build started: Project: TestRichie, Configuration: Debug Win32 ------
1>  TestRichie.cpp
1>d:\learning_library\umd\software architecture & design pattern\projects\proj#1\program\testrichie\testrichie\testrichie.cpp(84): warning C4715: 'AbstractRichie::GetConcreteRichie' : not all control paths return a v
1>  TestRichie.vcxproj -> D:\Learning_Library\UMD\software architecture & design pattern\projects\PROJ#1\program\TestRichie\Debug\TestRichie.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

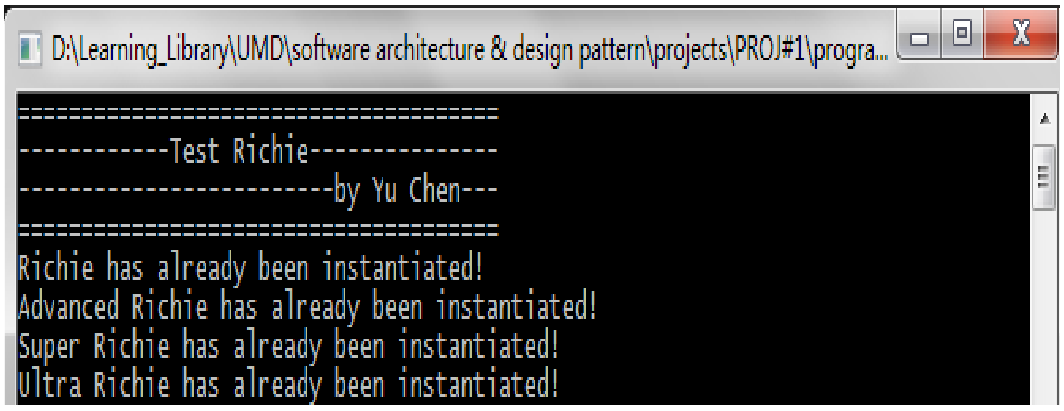
- Execution output
  - ✓ Test abstract factory



D:\Learning_Library\UMD\software architecture & design pattern\projects\PROJ#1\progra...

```
======================================
------------Test Richie---------------
-------------------------by Yu Chen---
======================================
Richie CPU
Richie MMU
Richie Motherboard
Advanced Richie CPU
Advanced Richie MMU
Advanced Richie Motherboard
Super Richie CPU
Super Richie MMU
Super Richie Motherboard
Ultra Richie CPU
Ultra Richie MMU
Ultra Richie Motherboard
```

- ✓ Test Singleton


D:\Learning_Library\UMD\software architecture & design pattern\projects\PROJ#1\progra...

```
========================================
-----------Test Richie--------------
------------------------by Yu Chen---
========================================
Richie has already been instantiated!
Advanced Richie has already been instantiated!
Super Richie has already been instantiated!
Ultra Richie has already been instantiated!
```

10. Final Status

It can test each component of each generation of computers and the instance of each generation of computers is unique.