Testing Object-Oriented Code: Unit Testing with Google Test and Catch2

Unit testing is an essential practice in software development, especially for object-oriented programming (OOP) where classes and their interactions form the core of the application. Two popular frameworks for unit testing in C++ are **Google Test** and **Catch2**. These frameworks provide simple ways to test object-oriented code, ensuring that each class behaves as expected and integrates well with others.

In this article, we'll cover the basics of unit testing OOP code with both Google Test and Catch2, explaining how to set them up and providing examples to show their practical use.

---

1. Google Test

Introduction to Google Test

Google Test is a widely used C++ testing framework developed by Google. It supports a wide variety of assertions, fixtures, and test cases, making it ideal for testing complex applications.

Setting Up Google Test

1. **Install Google Test:** To install Google Test, clone its repository and build it:

    ```
    git clone https://github.com/google/googletest.git
    cd googletest
    cmake .
    make
    ```

2. **Link Google Test:** Once built, link Google Test to your project. Add the following to your CMakeLists.txt:

```
find_package(GTest REQUIRED)
include_directories(${GTEST_INCLUDE_DIRS})

add_executable(your_test your_test.cpp)
target_link_libraries(your_test ${GTEST_LIBRARIES} pthread)
```

Example: Unit Testing with Google Test

Let's assume we have a simple class `Calculator`:

```
class Calculator {
public:
    int add(int a, int b) { return a + b; }
    int subtract(int a, int b) { return a - b; }
};
```

Now, we will write unit tests for the `Calculator` class using Google Test.

```
#include <gtest/gtest.h>
#include "calculator.h"

// Test case for addition
TEST(CalculatorTest, Addition) {
    Calculator calc;
    EXPECT_EQ(calc.add(2, 3), 5);
    EXPECT_EQ(calc.add(-1, -1), -2);
}

// Test case for subtraction
TEST(CalculatorTest, Subtraction) {
    Calculator calc;
    EXPECT_EQ(calc.subtract(5, 3), 2);
    EXPECT_EQ(calc.subtract(0, 0), 0);
}

// Main function to run the tests
int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

In this example:

- **`EXPECT_EQ`** asserts that the two values are equal. If they are not, the test will fail.
- Tests are grouped into the test suite (e.g., `CalculatorTest`), with each individual test (e.g., `Addition`, `Subtraction`) verifying different behaviors.

To run the tests:

```
./your_test
```

Google Test will output whether the tests passed or failed.

---

2. Catch2

Introduction to Catch2

Catch2 is another popular testing framework for C++. It is header-only, meaning you can integrate it directly into your codebase without needing to build any external libraries. Catch2 is also known for its simplicity and ease of use.

Setting Up Catch2

1. **Install Catch2:** You can install Catch2 by downloading the header file directly or using package managers like `vcpkg`.

   To install via `vcpkg`:

   ```
   ./vcpkg install catch2
   ```

2. **Include Catch2 in Your Project:** In your test file, include the Catch2 header:

   ```
   #define CATCH_CONFIG_MAIN
   #include <catch2/catch.hpp>
   ```

Example: Unit Testing with Catch2

Using the same `Calculator` class, we'll write unit tests using Catch2.

```
#define CATCH_CONFIG_MAIN
#include <catch2/catch.hpp>
#include "calculator.h"

// Test case for addition
TEST_CASE("Addition works correctly", "[calculator]") {
    Calculator calc;

    REQUIRE(calc.add(2, 3) == 5);
    REQUIRE(calc.add(-1, -1) == -2);
}

// Test case for subtraction
TEST_CASE("Subtraction works correctly", "[calculator]") {
    Calculator calc;

    REQUIRE(calc.subtract(5, 3) == 2);
    REQUIRE(calc.subtract(0, 0) == 0);
}
```

In this example:

- `REQUIRE` asserts that a condition is true, similar to `EXPECT_EQ` in Google Test.
- Test cases are defined using `TEST_CASE`, and you can tag them with categories (like `[calculator]`).

To run the tests:

```
./your_test
```

Catch2 provides detailed output, including which assertions failed and why.

Comparison: Google Test vs. Catch2

Feature	Google Test	Catch2
Installation	Requires external library	Header-only
Assertions	Rich set of assertions	Simpler assertion mechanism
Test Case Structure	TEST, TEST_F (for fixtures)	TEST_CASE
Output Format	Detailed, customizable output	Simple, easy-to-read output
Documentation	Well-documented, extensive	Easy to follow and concise

Conclusion

Both **Google Test** and **Catch2** provide excellent support for unit testing object-oriented code in C++. Google Test offers more extensive features, while Catch2 is simpler and easier to integrate into smaller projects. By writing tests for your classes, you can ensure that your object-oriented code behaves correctly, helping to catch bugs early and maintain a high level of code quality.

Testing is essential to any robust software development process, and both of these frameworks provide powerful ways to write and maintain your tests. Whether you are testing simple classes or more complex interactions between objects, unit testing will help you ensure the integrity of your codebase.