CS31 Project 5 Report

Benjamin He

UID: 804962948

**a. Notable Obstacles**

The only real obstacle I had with this project was getting the syntax for the class declarations and function implementations correct. For example, when I wanted to use the Kind enum in files other than AppleProduct.h, I needed to add AppleProduct:: in front of the enum so that the program recognized where the enum type was from. Also, I forgot to add semicolons after the closing curly brackets of both the class and the namespace scopes. Other than that, there wasn’t much trouble completing the project.

**b. Test Data**

I just pasted the code of the test cases used with a comment above each block of code to explain why I used it.

**//Test the default constructor and functions of AppleProduct class**

cs31::AppleProduct p;

assert(p.getCost() == 0.0);

assert(p.getKind() == cs31::AppleProduct::Kind::NOTSPECIFIED);

assert(p.getColor() == "");

assert(p.getVersion() == "");

**//Test constructor and class functions of AppleProduct class**

cs31::AppleProduct p1(cs31::AppleProduct::Kind::IPHONE, "8", "Space Grey", 699.00);

assert(std::to\_string(p1.getCost()) == "699.000000");

assert(p1.getKind() == cs31::AppleProduct::Kind::IPHONE);

assert(p1.getColor() == "Space Grey");

assert(p1.getVersion() == "8");

**//Test constructor and class functions of AppleProduct class**

cs31::AppleProduct p2(cs31::AppleProduct::Kind::IPAD, "8", "Space Grey", 429.00);

assert(std::to\_string(p2.getCost()) == "429.000000");

assert(p2.getKind() == cs31::AppleProduct::Kind::IPAD);

assert(p2.getColor() == "Space Grey");

assert(p2.getVersion() == "8");

**//test default constructor of AppleStore class**

cs31::AppleStore store;

**//test that the buyiPhone8 function with 64 gigabytes works correctly**

cs31::AppleProduct iphone8\_64 = store.buyiPhone8("Space Grey", 64);

assert(std::to\_string(iphone8\_64.getCost()) == "699.000000");

assert(iphone8\_64.getKind() == cs31::AppleProduct::Kind::IPHONE);

assert(iphone8\_64.getColor() == "Space Grey");

assert(iphone8\_64.getVersion() == "8");

**//test that the buyiPhone8 function with 256 gigabytes works correctly**

cs31::AppleProduct iphone8\_256 = store.buyiPhone8("Pink", 256);

assert(std::to\_string(iphone8\_256.getCost()) == "849.000000");

assert(iphone8\_256.getKind() == cs31::AppleProduct::Kind::IPHONE);

assert(iphone8\_256.getColor() == "Pink");

assert(iphone8\_256.getVersion() == "8");

**//test that the buyiPhone8Plus function with 64 gigabytes works correctly**

cs31::AppleProduct iphone8plus\_64 = store.buyiPhone8Plus("Blue", 64);

assert(std::to\_string(iphone8plus\_64.getCost()) == "799.000000");

assert(iphone8plus\_64.getKind() == cs31::AppleProduct::Kind::IPHONE);

assert(iphone8plus\_64.getColor() == "Blue");

assert(iphone8plus\_64.getVersion() == "8Plus");

**//test that the buyiPhone8 function with 256 gigabytes works correctly**

cs31::AppleProduct iphone8plus\_256 = store.buyiPhone8Plus("Red", 256);

assert(std::to\_string(iphone8plus\_256.getCost()) == "949.000000");

assert(iphone8plus\_256.getKind() == cs31::AppleProduct::Kind::IPHONE);

assert(iphone8plus\_256.getColor() == "Red");

assert(iphone8plus\_256.getVersion() == "8Plus");

**//test that the buyiPhoneX function with 64 gigabytes works correctly**

cs31::AppleProduct iphoneX\_64 = store.buyiPhoneX("White", 64);

assert(std::to\_string(iphoneX\_64.getCost()) == "999.000000");

assert(iphoneX\_64.getKind() == cs31::AppleProduct::Kind::IPHONE);

assert(iphoneX\_64.getColor() == "White");

assert(iphoneX\_64.getVersion() == "X");

**//test that the buyiPhoneX function with 256 gigabytes works correctly**

cs31::AppleProduct iphoneX\_256 = store.buyiPhoneX("Black", 256);

assert(std::to\_string(iphoneX\_256.getCost()) == "1149.000000");

assert(iphoneX\_256.getKind() == cs31::AppleProduct::Kind::IPHONE);

assert(iphoneX\_256.getColor() == "Black");

assert(iphoneX\_256.getVersion() == "X");

**//test that the buyiPad function with 32 gigabytes works correctly**

cs31::AppleProduct ipad\_32 = store.buyiPad("Turqoise", 32);

assert(std::to\_string(ipad\_32.getCost()) == "329.000000");

assert(ipad\_32.getKind() == cs31::AppleProduct::Kind::IPAD);

assert(ipad\_32.getColor() == "Turqoise");

assert(ipad\_32.getVersion() == "");

**//test that the buyiPad function with 128 gigabytes works correctly**

cs31::AppleProduct ipad\_128 = store.buyiPad("Olive", 128);

assert(std::to\_string(ipad\_128.getCost()) == "429.000000");

assert(ipad\_128.getKind() == cs31::AppleProduct::Kind::IPAD);

assert(ipad\_128.getColor() == "Olive");

assert(ipad\_128.getVersion() == "");

**//test that the buyiPadPro function with 64 gigabytes works correctly**

cs31::AppleProduct ipadPro\_64 = store.buyiPadPro("", 64);

assert(std::to\_string(ipadPro\_64.getCost()) == "649.000000");

assert(ipadPro\_64.getKind() == cs31::AppleProduct::Kind::IPAD);

assert(ipadPro\_64.getColor() == "");

assert(ipadPro\_64.getVersion() == "Pro");

**//test that the buyiPad function with 256 gigabytes works correctly**

cs31::AppleProduct ipadPro\_256 = store.buyiPadPro("Emerald", 256);

assert(std::to\_string(ipadPro\_256.getCost()) == "799.000000");

assert(ipadPro\_256.getKind() == cs31::AppleProduct::Kind::IPAD);

assert(ipadPro\_256.getColor() == "Emerald");

assert(ipadPro\_256.getVersion() == "Pro");

**//test that the buyiPad function with 512 gigabytes works correctly**

cs31::AppleProduct ipadPro\_512 = store.buyiPadPro("Maroon", 512);

assert(std::to\_string(ipadPro\_512.getCost()) == "999.000000");

assert(ipadPro\_512.getKind() == cs31::AppleProduct::Kind::IPAD);

assert(ipadPro\_512.getColor() == "Maroon");

assert(ipadPro\_512.getVersion() == "Pro");

**//test that the buyWatch1 function works correctly**

cs31::AppleProduct watch1 = store.buyWatch1("Silver");

assert(std::to\_string(watch1.getCost()) == "249.000000");

assert(watch1.getKind() == cs31::AppleProduct::Kind::WATCH);

assert(watch1.getColor() == "Silver");

assert(watch1.getVersion() == "1");

**//test that the buyWatch3 function works correctly**

cs31::AppleProduct watch3 = store.buyWatch3("Brown");

assert(std::to\_string(watch3.getCost()) == "329.000000");

assert(watch3.getKind() == cs31::AppleProduct::Kind::WATCH);

assert(watch3.getColor() == "Brown");

assert(watch3.getVersion() == "3");

**//make sure that std::logic\_error is thrown when gigabyte input is invalid //for buyiPhone8 function**

cs31::AppleStore s;

try

{

cs31::AppleProduct iphone8 = s.buyiPhone8("Space Grey", 9999);

assert(false);

}

catch (std::logic\_error) {

assert(true);

}

**//make sure that std::logic\_error is thrown when gigabyte input is invalid //for buyiPhone8Plus function**

try

{

cs31::AppleProduct iphone8plus = s.buyiPhone8Plus("Grey", 0);

assert(false);

}

catch (std::logic\_error) {

assert(true);

}

**//make sure that std::logic\_error is thrown when gigabyte input is invalid //for buyiPhoneX function**

try

{

cs31::AppleProduct iphoneX = s.buyiPhoneX("Purple", 32);

assert(false);

}

catch (std::logic\_error) {

assert(true);

}

**//make sure that std::logic\_error is thrown when gigabyte input is invalid //for buyiPad function**

try

{

cs31::AppleProduct ipad = s.buyiPad("Orange", 64);

assert(false);

}

catch (std::logic\_error) {

assert(true);

}

**//make sure that std::logic\_error is thrown when gigabyte input is invalid //for buyiPadPro function**

try

{

cs31::AppleProduct ipadPro = s.buyiPadPro("Yellow", 128);

assert(false);

}

catch (std::logic\_error) {

assert(true);

}

**//buyWatch1 and buyWatch3 don’t have exceptions to throw so no need to try catch them.**