Zane Koc

404984674

Cs31

Project 5 Report

The most notable obstacle I overcame was simply understanding the concept behind, and syntax of, object-oriented programming. Before the project I did not really understand what object-oriented even meant. With the help of the lecture slides and the internet I was able to understand what creating classes and objects meant and that helped make sense of the syntax because I actually knew what was going on. Also, related to this, the idea of namespaces was somewhat confusing. I now understand that they are essentially places where operations, classes etc. are stored and if a namespace is called you no longer must use its pre-fix before whatever operation is stored inside of it. Additionally, because this was the first project that we had to split the project into header and source files I had a hard time including the right things in each file, so they could call each other and everything be included in namespace cs31. Besides these two conceptual obstacles the project was straightforward, there were not any difficult pieces of logic to write into code.

**Test Data;**

**This tests if the watch1 member function works in conjunction with appleproduct**

cs31::AppleStore z;

cs31::AppleProduct watch1 = z.buyWatch1("blue");

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

assert(watch1.getCost() == 249.00);

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

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

**This tests if the watch3 member function works in conjunction with appleproduct**

cs31::AppleStore a;

cs31::AppleProduct watch3 = a.buyWatch3("blue");

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

assert(watch3.getCost() == 329.00);

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

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

**This tests if the ipad member function works in conjunction with appleproduct**

cs31::AppleStore b;

cs31::AppleProduct ipad = b.buyiPad("blue",32);

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

assert(ipad.getCost() == 329.00);

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

assert(ipad.getColor() == "blue");

cs31::AppleStore b;

cs31::AppleProduct ipad = b.buyiPad("blue",128);

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

assert(ipad.getCost() == 329.00);

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

assert(ipad.getColor() == "blue");

**This tests if the ipadpro member function works in conjunction with appleproduct**

cs31::AppleStore c;

cs31::AppleProduct ipadpro = b.buyiPadPro("blue", 64);

assert(ipadpro.getVersion() == "Pro");

assert(ipadpro.getCost() == 649.00);

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

assert(ipadpro.getColor() == "blue");

cs31::AppleStore c;

cs31::AppleProduct ipadpro = b.buyiPadPro("blue", 256);

assert(ipadpro.getVersion() == "Pro");

assert(ipadpro.getCost() == 649.00);

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

assert(ipadpro.getColor() == "blue");

cs31::AppleStore c;

cs31::AppleProduct ipadpro = b.buyiPadPro("blue", 512);

assert(ipadpro.getVersion() == "Pro");

assert(ipadpro.getCost() == 649.00);

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

assert(ipadpro.getColor() == "blue");

**This tests if the iphone8 member function works in conjunction with appleproduct**

cs31::AppleStore d;

cs31::AppleProduct iphone8 = b.buyiPhone8("blue", 64);

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

assert(iphone8.getCost() == 699.00);

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

assert(iphone8.getColor() == "blue");

cs31::AppleStore d;

cs31::AppleProduct iphone8 = b.buyiPhone8("blue", 256);

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

assert(iphone8.getCost() == 699.00);

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

assert(iphone8.getColor() == "blue");

**This tests if the iphone8plus member function works in conjunction with appleproduct**

cs31::AppleStore e;

cs31::AppleProduct iPhone8Plus = b.buyiPhone8Plus("blue", 64);

assert(iPhone8Plus.getVersion() == "8Plus");

assert(iPhone8Plus.getCost() == 799.00);

assert(iPhone8Plus.getKind() == cs31::AppleProduct::Kind::IPHONE);

assert(iPhone8Plus.getColor() == "blue");

cs31::AppleStore e;

cs31::AppleProduct iPhone8Plus = b.buyiPhone8Plus("blue", 256);

assert(iPhone8Plus.getVersion() == "8Plus");

assert(iPhone8Plus.getCost() == 799.00);

assert(iPhone8Plus.getKind() == cs31::AppleProduct::Kind::IPHONE);

assert(iPhone8Plus.getColor() == "blue");

**This tests if the iphoneX member function works in conjunction with appleproduct**

cs31::AppleStore f;

cs31::AppleProduct iPhoneX = b.buyiPhoneX("blue", 64);

assert(iPhoneX.getVersion() == "X");

assert(iPhoneX.getCost() == (999.00));

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

assert(iPhoneX.getColor() == "blue");

cs31::AppleStore f;

cs31::AppleProduct iPhoneX = b.buyiPhoneX("blue", 256);

assert(iPhoneX.getVersion() == "X");

assert(iPhoneX.getCost() == (999.00));

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

assert(iPhoneX.getColor() == "blue");

**This makes sure all the functions throw logic errors when they are supposed to**

try

{

cs31::AppleStore store;

cs31::AppleProduct iPad = store.buyiPad("black", 33);

cs31::AppleProduct iPadPro = store.buyiPadPro("black", 65);

cs31::AppleProduct iPhone8 = store.buyiPhone8("black", 65);

cs31::AppleProduct iPhone8Plus = store.buyiPhone8Plus("black", 65);

cs31::AppleProduct iPhoneX= store.buyiPhoneX("black", 65);

cerr << "trying..." << endl;

assert(false);

}

catch (logic\_error)

{

assert(true);

}

If this runs through without error it means both classes work, and work together, for all combinations of products and specs.