CS 2212

Programming Project 1 Class Application

I want **you and your partner** to create a program that works with vector and a class. It must be 100% clear to me who created each part of your system. You can break up the work is any reasonable way except to do everything together. You should show each other your code and get help, but **each partner should own an equal portion of the program. You will use this code as part of your final project.**

One partner can create the model class (header and source code files) and the other the controller program (source code file). It is also acceptable to break up the work by function or method. Each block comment for each method or function including main() should show the author's name. If the same author is responsible for the entire source code or header file, then they author may only be noted in the block comment at the top of your code file.

1 Goals

1. Create and use class in a C++ program.
2. Use the MVC Design pattern to design your class and controller program.
3. Create a model class made up of two separate files, one for the .h header file and another for the .cpp code file .
4. Create a properly designed controller program that uses the model class and is in its own source code .cpp file.
5. Create a vector to store the class objects.
6. Read data from the file and use it to create objects of the class.
7. Store the collection of objects in the vector.
8. Use the vector to manage the class objects and call their functions.
9. Save the data from the objects in the vector to a file. Make the data easy to read (clear text).

2 Requirements

Your task is to create a simple Object-Oriented program for using a simple class. Make up your own class based on something in real life or a game. The class cannot be a bank transaction, an ATM, Venmo, pets, items, or people in a contact list. It can be based on a transaction-based system for a game or real business.

Your program will have three compile modules – 2 for the object's model class (header .h and code .cpp) and the third for the controller program (another .cpp). This program will be expanded for your final project.

You should use the divide and conquer problem solving technique and MVC Pattern (you won't have a view) to create your project.

Don't forget to test your code to make sure that it works after each new small addition!

The outline of the **model class** (you can add to this but this all of these are needed):

a) **Get approval of your idea from me before you start, first come, first served. KEEP is simple.**

Programming Project 1 Class Application

b) Your class should have four to six data members (attributes). Do NOT store a collection in the object, this is too hard for your first use of a class.

c) The only common element is there must be a date in your object – DOB, transaction date, due date, any kind of date.

  i) It can be stored as text, numbers, using an enumerated type, or you can use the Chrono library available in C++. https://en.cppreference.com/w/cpp/chrono

d) **You need a minimum of 2 other data members of the class besides those related to the date. At least one string and a number.**

e) **Class methods. Remember** – the class methods are doing something to one instance/item/object of the class.

f) Your model class must have two constructors.

  i) One is a "no argument" constructor which creates a default or empty object.

  ii) The second is a constructor with a parameter for each attribute to allow you to create a new object from data in a file or user entry.

g) **There should be a isDate() method that** returns a boolean value. The method should accept an argument of a date. The method needs to check to see if the object's date is the same as the date that was passed as an argument. If the date is the same as the object's date return true, otherwise false.

h) **There should be a toString() method that** returns a string containing the data that you could use to print the object to the console.

i) **There should be a toFile() method** that returns a string containing the data formatted for you to store it in a file (make it easy to read back when you start-up your program). You can use a delimiter.

j) **There should be a method** that *has a parameter that is used by the method* to update something. For example, a player can do an action that results in their loss or gain of points in a game (action), mark something done or cancelled (update status), etc. Be careful, do not create a mutator method. The data should be used by the method to decide what needs to change.

k) You class cannot have any mutators at all. Do not allow direct access to change your class data.

l) Your class can have an accessor (getter) for one attribute. If you find yourself needing a lot of accessors, you have a design flaw and need to move the action you are doing into the class.

m) Your class can have additional methods that do necessary work, but not too many (keep it simple).

Controller program data and functions and the model class data members and functions.
You need to carefully assign responsibilities to the Model Class or the Controller using the MVC design pattern we discussed. We also discussed how the model class would work with the controller.

You need to **create a controller program** that **uses your model class**. In your controller program:

1. You need to use the vector template class to create a vector of the model class that will contain the collection of model class objects. E.g `vector<myClass> mydata;`

Programming Project 1 Class Application

2. Read the plain text data from the file (call it **data.txt**) and use the model class constructor with parameters to create an object for each row in the file and add it to the collection in the vector.
    a. The first time you run the program there will not be any data in the file (in fact the file may not exist yet).
    b. Make sure your program handles this properly and continues running.
    c. If the file is empty or missing, you should send a message saying there is not data or no file, but then your program must continue.
3. After you read the data from the file, you should enter a large loop with a menu where the user can choose the next action and has the option to quit. This loop will run everything else in the program until the user is done.
4. When the user is done, the data in the vector should be saved to the file as clear text and the program should end gracefully.
5. The options on the menu should allow the user to choose from the following:
    a. Add a new object.
        i. You will have to use console input to get the data for the new objects in your controller program. Then use the class constructor with the data as arguments and add the resulting object to the vector.
        ii. You can choose to do the challenge and create a readData() method in your class to handle the dialog to get the user input of data.
    b. Search for objects with a certain date and print them out in a list.
        i. This will require that you use a loop to check each item in the vector to see if it has the date you are looking for with the isDate() method from your class.
        ii. If the date matches, use the toString() method to print the object out.
    c. Print all the objects in a table to the console.
        i. Your controller program should use a loop to print all the data in the vector out in a nice, neat format (table like). You should use toString() method from the model class when you print the data in the vector.
    d. Create a menu item that allows the user to choose each of the methods you created.
    e. Your controller program should use all the methods in your model class at least once.
    f. The menu should have a way to quit.
6. When the user quits, make sure your program uses a loop to save the data in the collection to a file (one row per object using the toFile() method from your model class). This is the file you will read the data from when you start-up the program.
7. End your program gracefully.

Follow the style guide.

8. Write clear and informative prompts and welcome/end of program output.
9. You should be validating your input, and invalid input should not crash the program.
10. Make sure you have good comments especially your block comments.

Programming Project 1 Class Application

11. Make sure your data printout has headers showing what each field is and make the data line up and look nice.
12. Make sure program ends gracefully for all situations.
13. Be customer focused and see me with questions about my requirements at least once.
14. Make sure you are creating customer value; make a program your friends want. Think about how can you add value?

Challenges for extra credit (up to +2):

- Instead of collecting the data for the model object in your controller program, delegate this to your model class and add the object to your collection. You can do this by using your no-argument constructor to do this.
  - To do this, you need to create a readData() method in the model class that has a dialog to collect and update the data of the object.

3 Testing Design and Output.

Create a written test plan with test cases that examine all these test scenarios and any others you deem necessary. Create a sequence of test cases or plans of what you will enter so that you test your program fully. **Save the console output from each of these tests and put them in a text file called output.txt and hand it in.**

Test your program by doing the following:

- Use white box testing to make sure all the code in your program is executed with all possible cases (every selection statement is tested where it is true and false)
- Black box testing to test each requirement.
- Think about validation and fuzzing test cases. Invalid input should not get through but should not crash the program.
- Test without a data file.
- Test with an empty data file.
- Exit before you add any objects via console.
- Do everything right, create at least 10 model class objects and put them to the collection (the vector) and save them to the file and read them from the file. Print them out.
- Search for dates that at least one object has. Do several in a row.
- Search for dates that none of the objects have.
- Search for a date one day before and one day after the date at least one object has.
- Search for a date that 2 or more objects have.
- Make some errors in input when entering data for a new object, choosing a menu item, or entering a date.
- Exit before you test each function.
- When related to console input, send correct and incorrect arguments to each function that has arguments.

Programming Project 1 Class Application

TEST PLAN and Output Example:
********************* Test 1 – No file, quit before you do anything ***********************
// console output from this test goes here.

******************* Test 2 – No file, enter 10 objects and print then quit ******************
//console output from this test goes here

***** Test 2 – File has 10 rows, enter 2 more objects, search for April 10, print then quit *****
//console output from this test goes here

3 Submit via Canvas

Each block comment for each method or function including main() **should show the author's name.** If the same author is responsible for the entire source code or header file, then they author should be noted in the block comment at the top of your program.

Only source code and header files and text files should be submitted. No compiled or executable code. Submit your program and files into the Canvas Assignment.

Hand in the following files and nothing else:

a. A description of what your program is supposed to do and a description of your class.
b. The header file (.hpp) and source code(.cpp) for your model class.
c. The .cpp source code for your controller program, you can call it main.cpp.
d. Your written test plan and the console output for all your test runs, clearly labeled in a .txt file.

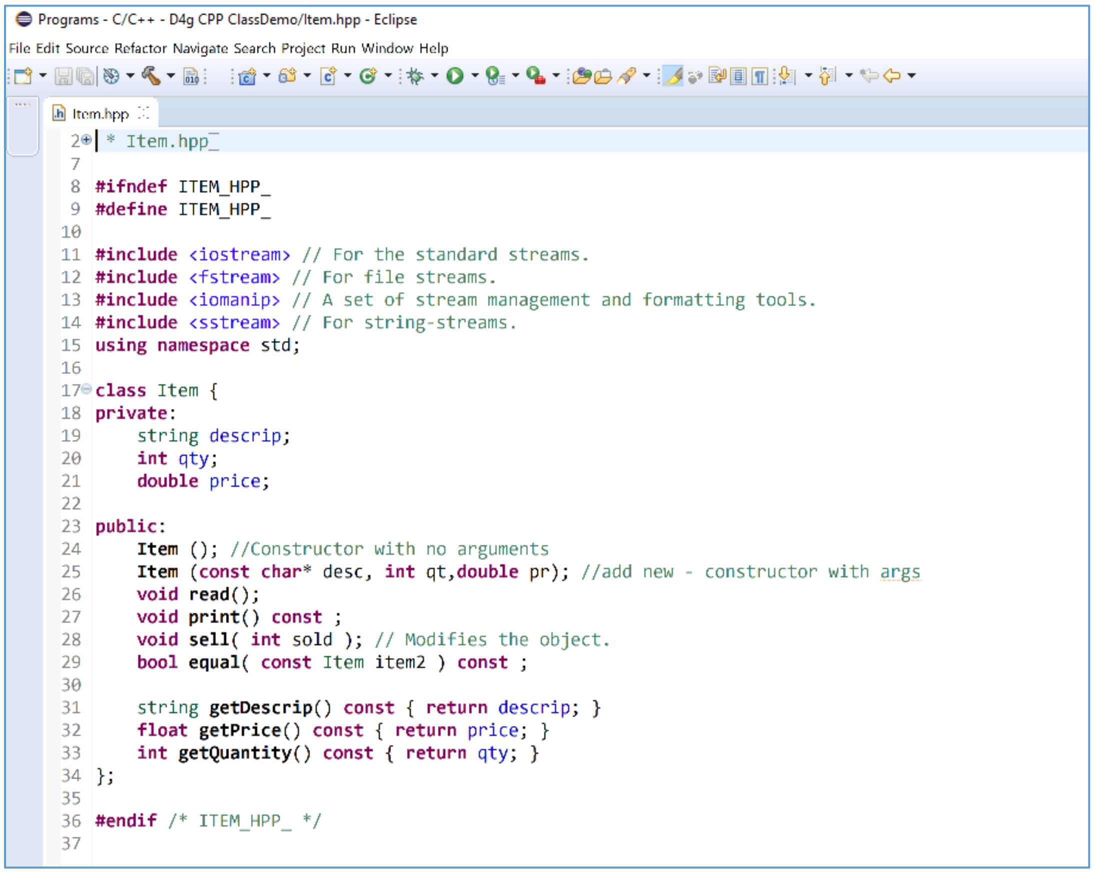
You have 2 weeks to do this project, see the deadline in Canvas. **You will use this code as part of your final project, so following the requirements exactly is important.**

For help with your program, come to my office hours or see me or our class TA.

Programming Project 1 Class Application

Here is an example of how to do some basic things in a MVC system using a simple class. Trying to apply this exactly for your project will not work, there are important differences.

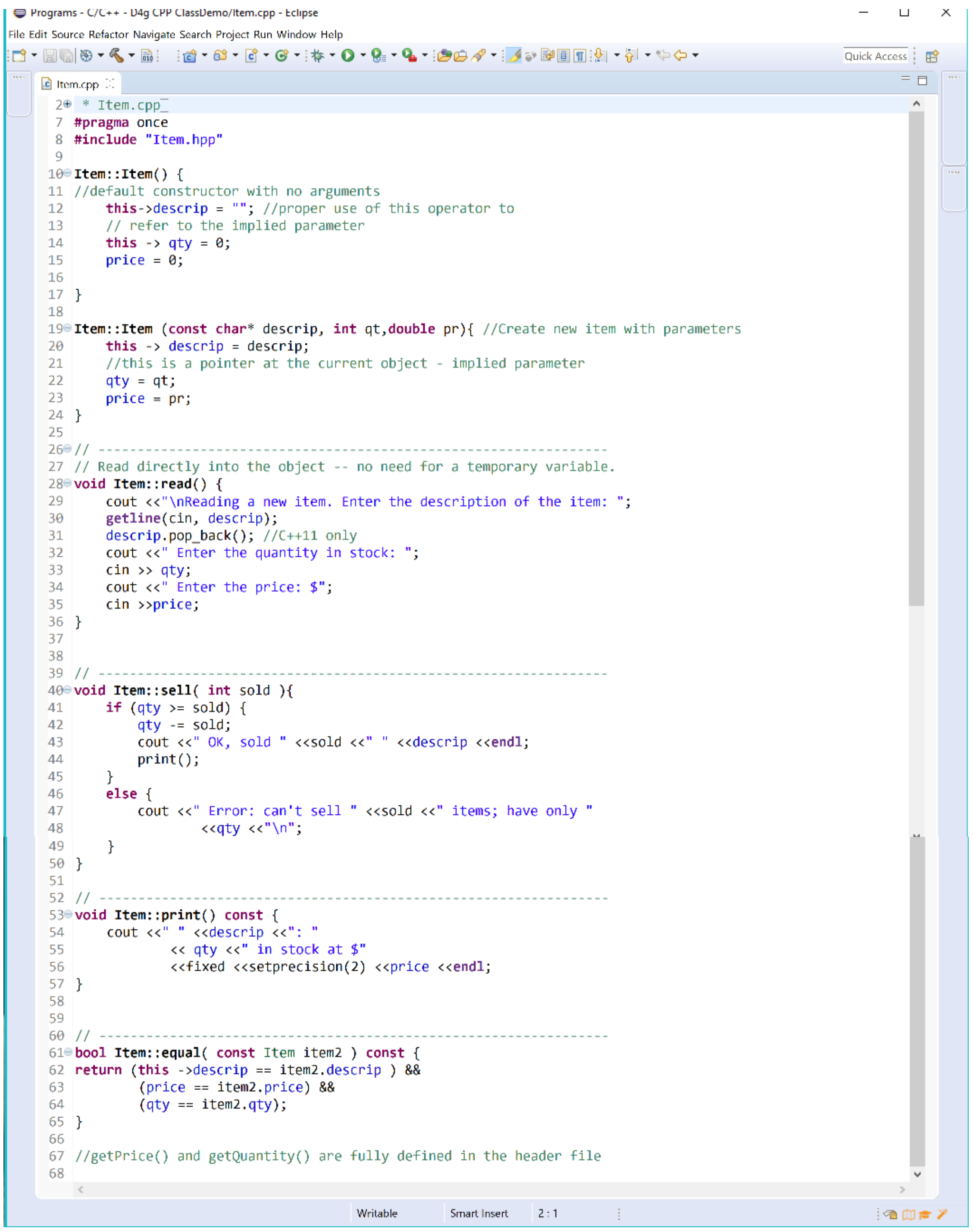
Model class header file example item.hpp


Programs - C/C++ - D4g CPP ClassDemo/Item.hpp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Item.hpp

```
 * Item.hpp_

#ifndef ITEM_HPP_
#define ITEM_HPP_

#include <iostream> // For the standard streams.
#include <fstream> // For file streams.
#include <iomanip> // A set of stream management and formatting tools.
#include <sstream> // For string-streams.
using namespace std;

class Item {
private:
    string descrip;
    int qty;
    double price;

public:
    Item (); //Constructor with no arguments
    Item (const char* desc, int qt,double pr); //add new - constructor with args
    void read();
    void print() const ;
    void sell( int sold ); // Modifies the object.
    bool equal( const Item item2 ) const ;

    string getDescrip() const { return descrip; }
    float getPrice() const { return price; }
    int getQuantity() const { return qty; }
};

#endif /* ITEM_HPP_ */
```

Programming Project 1 Class Application

Source code for model class example, item.cpp



Programs - C/C++ - D4g CPP ClassDemo/Item.cpp - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

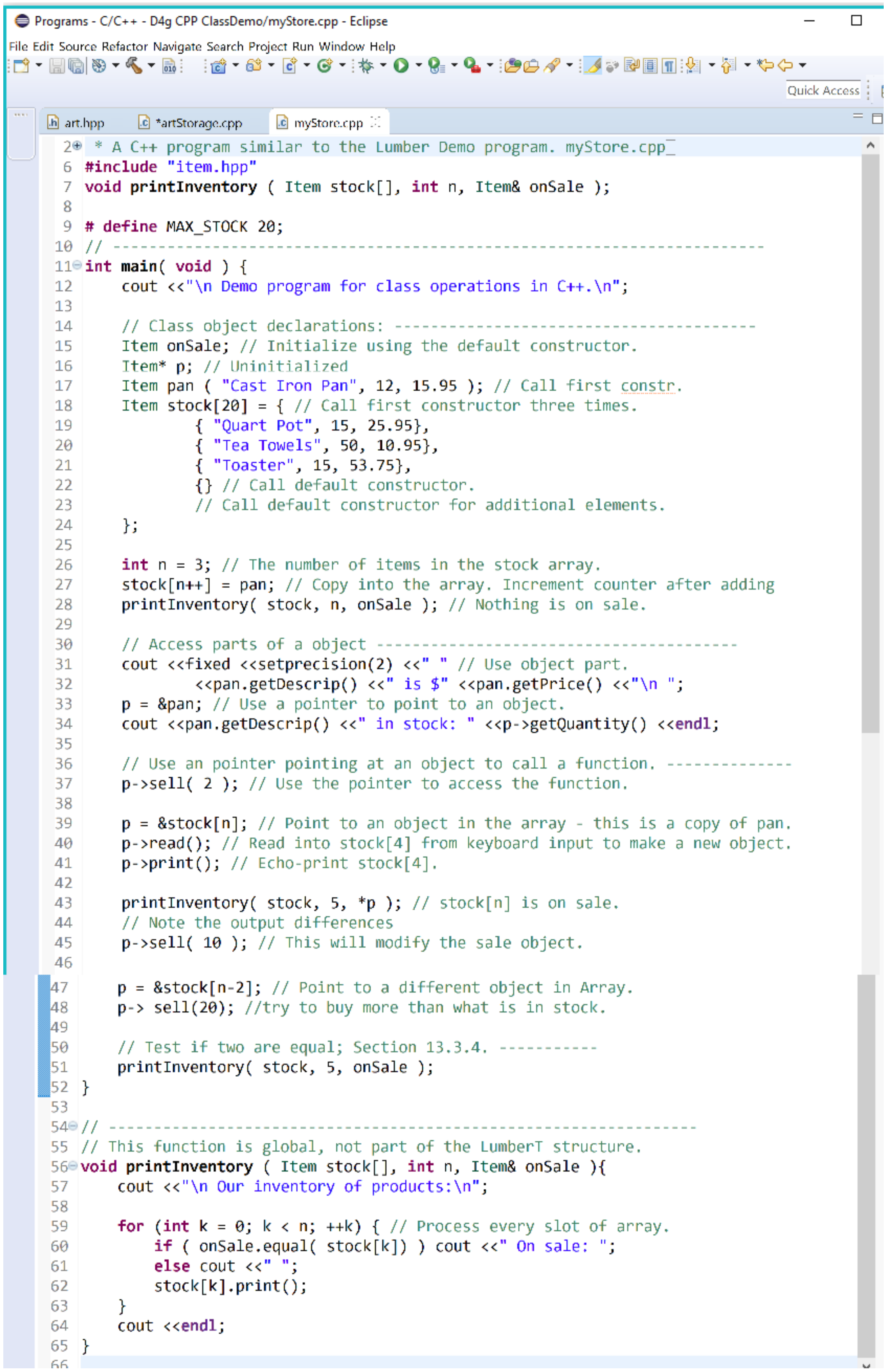
Item.cpp

```cpp
 * Item.cpp
#pragma once
#include "Item.hpp"

Item::Item() {
//default constructor with no arguments
    this->descrip = ""; //proper use of this operator to
    // refer to the implied parameter
    this -> qty = 0;
    price = 0;

}

Item::Item (const char* descrip, int qt,double pr){ //Create new item with parameters
    this -> descrip = descrip;
    //this is a pointer at the current object - implied parameter
    qty = qt;
    price = pr;
}

// ----------------------------------------------------------------
// Read directly into the object -- no need for a temporary variable.
void Item::read() {
    cout <<"\nReading a new item. Enter the description of the item: ";
    getline(cin, descrip);
    descrip.pop_back(); //C++11 only
    cout <<" Enter the quantity in stock: ";
    cin >> qty;
    cout <<" Enter the price: $";
    cin >>price;
}


// ----------------------------------------------------------------
void Item::sell( int sold ){
    if (qty >= sold) {
        qty -= sold;
        cout <<" OK, sold " <<sold <<" " <<descrip <<endl;
        print();
    }
    else {
        cout <<" Error: can't sell " <<sold <<" items; have only "
                <<qty <<"\n";
    }
}

// ----------------------------------------------------------------
void Item::print() const {
    cout <<" " <<descrip <<": "
            << qty <<" in stock at $"
            <<fixed <<setprecision(2) <<price <<endl;
}


// ----------------------------------------------------------------
bool Item::equal( const Item item2 ) const {
return (this ->descrip == item2.descrip ) &&
        (price == item2.price) &&
        (qty == item2.qty);
}

//getPrice() and getQuantity() are fully defined in the header file

```

Writable   Smart Insert   2 : 1

Programming Project 1 Class Application

Source code for controller class example myStore.cpp


Programs - C/C++ - D4g CPP ClassDemo/myStore.cpp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
art.hpp
*artStorage.cpp
myStore.cpp

```
 * A C++ program similar to the Lumber Demo program. myStore.cpp_
#include "item.hpp"
void printInventory ( Item stock[], int n, Item& onSale );

# define MAX_STOCK 20;
// ------------------------------------------------------------------------
int main( void ) {
    cout <<"\n Demo program for class operations in C++.\n";

    // Class object declarations: ----------------------------------------
    Item onSale; // Initialize using the default constructor.
    Item* p; // Uninitialized
    Item pan ( "Cast Iron Pan", 12, 15.95 ); // Call first constr.
    Item stock[20] = { // Call first constructor three times.
            { "Quart Pot", 15, 25.95},
            { "Tea Towels", 50, 10.95},
            { "Toaster", 15, 53.75},
            {} // Call default constructor.
            // Call default constructor for additional elements.
    };

    int n = 3; // The number of items in the stock array.
    stock[n++] = pan; // Copy into the array. Increment counter after adding
    printInventory( stock, n, onSale ); // Nothing is on sale.

    // Access parts of a object ---------------------------------------
    cout <<fixed <<setprecision(2) <<" " // Use object part.
            <<pan.getDescrip() <<" is $" <<pan.getPrice() <<"\n ";
    p = &pan; // Use a pointer to point to an object.
    cout <<pan.getDescrip() <<" in stock: " <<p->getQuantity() <<endl;

    // Use an pointer pointing at an object to call a function. -------------
    p->sell( 2 ); // Use the pointer to access the function.

    p = &stock[n]; // Point to an object in the array - this is a copy of pan.
    p->read(); // Read into stock[4] from keyboard input to make a new object.
    p->print(); // Echo-print stock[4].

    printInventory( stock, 5, *p ); // stock[n] is on sale.
    // Note the output differences
    p->sell( 10 ); // This will modify the sale object.

    p = &stock[n-2]; // Point to a different object in Array.
    p-> sell(20); //try to buy more than what is in stock.

    // Test if two are equal; Section 13.3.4. -----------
    printInventory( stock, 5, onSale );
}

// ----------------------------------------------------------------
// This function is global, not part of the LumberT structure.
void printInventory ( Item stock[], int n, Item& onSale ){
    cout <<"\n Our inventory of products:\n";

    for (int k = 0; k < n; ++k) { // Process every slot of array.
        if ( onSale.equal( stock[k]) ) cout <<" On sale: ";
        else cout <<" ";
        stock[k].print();
    }
    cout <<endl;
}
```