C++ Version

InventoryItem class definition:

* Private member variables:
  + String variable that holds a description of the item (called description)
  + Double variable that holds the item’s cost (called cost)
  + Int variable that holds the number of units on hand for each item (called units)
* Public member functions, constructors, and destructor
  + Constructor #1 (default constructor): sets arbitrary values to each object that is created
  + Constructor #2 (overloaded constructor):
    - one parameter: a string whose value is assigned to the description member variable
    - sets arbitrary values for item cost and number of units
  + Constructor #3 (overloaded constructor):
    - Three parameters:
      * A string whose value will be assigned to the description member variable
      * A double whose value will be assigned to the cost member variable
      * An int whose value will be assigned to the units member variable
  + Mutator functions:
    - A void function called setDescription that takes one string parameter and assigns its value to the description member variable.
    - A void function called setCost that takes one double parameter and assigns its value to the cost member variable.
    - A void function called setUnits that takes one int parameter and assigns its value to the units member variable.
  + Accessor functions:
    - A function called getDescription that returns the value of the description member variable
    - A function called getCost that returns the value of the cost member variable
    - A function called getUnits that returns the value of the units member variable

Register class definition:

* Private member variables:
  + An array of 4 InventoryItem objects called inventory
  + An int variable that holds the index of the item called itemindex
  + A double variable that holds the subtotal of the purchase called subtotal
  + A double variable that holds the tax called tax
  + A double variable that holds the total of the purchase called purchasetotal
* Public member functions, constructors, and destructor;
  + Constructor #1 (default constructor): creates the inventory array
    - Contains 4 instances of the InventoryItem class that are defined with the 3rd InventoryItem overload constructor
  + Function to find and change the item quantity called itemquantity (Takes no parameters and returns nothing)
    - Summary: this member function allows the user to decide which item they would like to purchase and the quantity. It includes user validation and repeats until for the user to add as many items to their purchase as they would like
    - Algorithm:
    - The function should continue asking the user if they want to n=buy something else until they indicate that they don’t want to
    - Create a loop that will continue as long as the user hasn’t selected a valid item to purchase
    - Display the quantities of each item from the inventory
      * Use a for loop to loop through each element of the inventory array
      * Use the getUnits method to find the quantity of the current item, and the getDescription item to find the name of the current item
    - Ask the user which item they want to purchase (and store their answer in a string variable)
    - Check to see whether the item is actually in the store
      * Change each letter to lowercase (if it isn’t already) by looping through each character in the string and using the tolower method
      * Loop through the inventory array to try and match the lowercase input the description of one of the array elements
        + If the item has a match, then save the index of the item found in inventory to be used later
        + If not then repeat the loop (asking the user what they want to buy)
    - Ask the user how much they want of each item (this should loop as long as the user’s input is invalid)
      * Save input as a string named quant
      * If the string only contains an integer (check using isdigit method) then convert the string to an integer and save it in a variable named quantity, and indicate that the input was valid so the loop will exit
      * If not then indicate that the user’s input was invalid so the loop will run again
    - Check to see if the quantity the user asked for is in stock (this should continue to ask again for the user’s desired quantity and loop as long as the quantity the user asked for isn’t available)
      * If the quantity of the item (found using the getUnits method) is available, then the item’s quantity (in the inventory array) is updated using the setUnits method (with the argument being the user’s desired quantity subtracted from the quantity of the item that is in stock)
        + The order is then added to a map containing the items the user wants to purchase mapped to the quantities of each of the items
      * If the quantity of the item is too large or negative, then an error will be displayed and the user will be asked to input a different quantity
  + Function to calculate subtotal called subtotal1 (Takes no parameters and returns nothing)
    - Loops through each item in the inventory array and checks it against the map of items ordered (called order)
      * If the quantity of the item in the order map is not 0, then the cost of the item is found and then multiplied by 1.3 (to get the profit of the item)
      * Update the purchase subtotal member variable by adding the result from the last step
    - Find the tax on the purchase by multiplying the subtotal by 0.06 and save this value in the tax member variable
    - Find the purchase total by adding the tax and subtotal and save this value in the purchse member variable
  + Function to display the purchase total called display (Takes no parameters and returns nothing)
    - Print each item in a list
      * Loops through the inventory array and order map to find each purchase and uses the index of the map to print the item quantity and getDescription method to print the name of the item
    - Print the subtotal, tax, and purchase

Main function (user interface)

* The main function will repeat as long as the user indicates that they wish to do so
  + A do-while loop should be used, with the condition being an answer of “yes” when asked if the user wants to continue
    - This response should be stored outside of the loop as a string variable
* a Register object to hold all information from the purchase should be created
  + the item should be created outside of the do-while loop so that the quantities of each item don’t reset with each new purchase
* ask the user which items they want to buy and the different quantity
  + call the itemquantity function
* after the user has indicated that they are finished choosing items, find the subtotal of the user’s purchase
  + call the subtotal1 function
* display the user’s purchases
  + call the display function
* ask the user if they want to start a new order
  + this response will be used as the condition for the do-while loop
  + the loop will continue only if the user chooses yes

Ruby Version

InventoryItem class definition:

* Private member variables:
  + variable that holds a description of the item (called description)
  + variable that holds the item’s cost (called cost)
  + variable that holds the number of units on hand for each item (called units)
* Public member functions, constructors, and destructor
  + Constructor #1 (default constructor): sets arbitrary values to each object that is created
  + Constructor #2 (overloaded constructor):
    - one parameter: a string whose value is assigned to the description member variable
    - sets arbitrary values for item cost and number of units
  + Constructor #3 (overloaded constructor):
    - Three parameters:
      * A string whose value will be assigned to the description member variable
      * A double whose value will be assigned to the cost member variable
      * An int whose value will be assigned to the units member variable
  + Mutator functions:
    - A function called setDescription that takes one string parameter and assigns its value to the description member variable.
    - A function called setCost that takes one double parameter and assigns its value to the cost member variable.
    - A function called setUnits that takes one int parameter and assigns its value to the units member variable.
  + Accessor functions:
    - A function called getDescription that returns the value of the description member variable
    - A function called getCost that returns the value of the cost member variable
    - A function called getUnits that returns the value of the units member variable

Register class definition:

* Private member variables:
  + An array called inventory that will hold the store’s inventory
  + A variable that holds the referencing index of the item called itemindex
  + A variable that holds the subtotal of the purchase called subtotal
  + A variable that holds the tax called tax
  + A variable that holds the total of the purchase called purchasetotal
  + A Hash variable called order
* Public member functions, constructors, and destructor;
  + Constructor #1 (default constructor): creates the inventory array
    - Create 4 instances of the InventoryItem class that are defined with the 3rd InventoryItem overload constructor and adds them to the array
    - Create a hash (called order) that has the index of the item in the array as the key and the quantity that the user wants of that item as the value
  + Function to display the store's inventory called displayitems
    - uses the array class's each function to go to each element in the array and display it to the user
  + Function that checks whether an item is in the inventory called validitem that takes one argument
    - put the item variable into lowercase
    - loops through each element of the array
      * checks to see if the item is in the inventory using the getDescription function
      * if so: checks whether the quantity is 0 using the getUnits method so that user can't buy an item that's out of stock, if not repeats and asks again
      * saves the valid item in the itemindex member variable
  + Function to chekc whether the quantity input is valid that takes 1 argument
    - checks whether the argument is an integer (if yes returns true, otherwise false)
  + Function to check whether the quantity of an item is available that takes one argument
    - if the requested quantity is greater than 10 or greater than the number of units available (found using getUnits method on the item) then the function returns false, otherwise returns true
  + function to update the inventory called updateinvent
    - updates rhe quantity of the item in the inventory using the setUnits member function
  + function to display the current order called displayorder
    - loops through each element in the inventory array and displays the order if the user has purchased the item
    - displays the subtotal so far
  + function that prompts the user for the name of an item called itemorder
    - calls the displayitems function
    - create a do while loop that exits if the item is found in the inventory
    - uses the user's input as the argument for the validitem method
  + Function called quant that prompts user for the quantity
    - do-while loop that continues if the quantity is unavailable (converts user input to an int check using itemavail method with the user input as the argument)
    - do-while loop that continues if the user input isn't an integer (checks using validquantity with the user input as the argument)
  + Function to find and change the item quantity called itemquantity (Takes no parameters and returns nothing)
    - do-while loop that ends if the user indicates they want to leave
    - calls the itemorder function so the user can input which item they want
    - calls the quant function so the user can input the quantity of the item that they want
    - calls the displayorder function to display the order so far
    - ask the user if they want to leave and uses input as a condition in the do while loop
  + Function to calculate subtotal called subtotal1 (Takes no parameters and returns nothing)
    - Loops through each item in the inventory array and checks it against the map of items ordered (called order)
      * If the quantity of the item in the order map is not 0, then the cost of the item is found and then multiplied by 1.3 (to get the profit of the item)
      * Update the purchase subtotal member variable by adding the result from the last step
    - Find the tax on the purchase by multiplying the subtotal by 0.06 and save this value in the tax member variable
    - Find the purchase total by adding the tax and subtotal and save this value in the purchase member variable
    - Round tax, subtotal, and purchase to 2 decimal places
  + Function to display the purchase total called display (Takes no parameters and returns nothing)
    - Print each item that the user purchased in a list
      * Loops through the inventory array and order map to find each purchase and uses the index of the map to print the item quantity and getDescription method to print the name of the item
    - Print the subtotal, tax, and purchase
    - Clears the hash by setting each index’s value to 0

Main function (user interface)

* The main function will repeat as long as the user indicates that they wish to do so
  + A do-while loop should be used, with the condition for exiting being an answer of “no” when asked if the user wants to continue
    - This response should be stored outside of the loop as a string variable
* a Register object to hold all information from the purchase should be created
  + the item should be created outside of the do-while loop so that the quantities of each item don’t reset with each new purchase
* ask the user which items they want to buy and the different quantity
  + call the itemquantity function
* after the user has indicated that they are finished choosing items, find the subtotal of the user’s purchase
  + call the subtotal1 function
* display the user’s purchases
  + call the display function
* ask the user if they want to start a new order
  + this response will be used as the condition for the do-while loop