2023

# CP6200 JavaProgramming2 OER - OOP assignment - Item and Shopping Cart classes

Shoshana Marcus
*CUNY Kingsborough Community College*

Kingsborough Computer College, CUNY
Department of Mathematics and Computer Science
Dr. Shoshana Marcus

CP 6200: Java Programming II
Introduction to Object Oriented Programming

Programming Assignment

**Objective:** In this assignment you will get comfortable creating your own classes in Java. In the first phase, you will create a class for the first time and use it in a driver program. In the second phase, you will enhance the class and use it in an array of objects. In the third phase, you will create a new class that uses aggregation (or composition) of objects.

## Phase 1:

Design a class that stores information about products that are sold in a store. Write a class named `Item` that has (at least) the following member variables:

- **name**: a `String` that holds the name of the inventory item.
- **vendor**: a `String` that stores the name of the vendor for the product.
- **price**: a `double` that stores the current selling price of the item (how much the customer pays to buy it).
- **cost**: a `double` that stores the current cost price of the item (how much the store pays to acquire it).
- **weight**: a `double` that stores the weight of a single item of this type.
- **taxable**: a `boolean` field that indicates whether tax is charged on the item.

Make sure to choose appropriate access specifiers for the data members and methods in your `Item` class!

In addition, the class should have the following member methods:

- **Constructor.** The constructor should accept the item's name, cost, and price as arguments and assign these values to the object's `name`, `cost`, and `price` member variables. The constructor should initialize the `weight` to 1 and `taxable` to `true`.
- **Accessors.** Appropriate accessor methods should be created to allow values to be retrieved from an object's `name, vendor, price, weight`, and `taxable` member variables.
- **Mutators.** Appropriate mutator methods should be created to allow values to be changed in an object's `weight` and `taxable` fields.
- **increaseCost.** This method should increase the cost price by 5% (to account for inflation). This is a `void` method since it modifies the current state of the object.

- **profit.** This method should accept no parameters and return the profit on the item, which is calculated as the cost subtracted from the price.

**Driver Program**

Demonstrate the class in a program that creates an `Item` object. Then increase the cost 3 times (by 5% each time), calculate the profit and display it on the screen. Call the mutator method that sets the weight to a number you specify. Create several other `Item` objects and see that the fields have different values.

**Feel free to use your imagination to enhance the Item class!**

**Sample code**

If your `Item` class is written properly, the following lines of code should be valid in your main method:

```
Item chair = new Item("Desk Chair", 30, 55);
//increase cost 3 times, due to inflation.
chair.increaseCost();
chair.increaseCost();
chair.increaseCost();
//display the profit
System.out.println("The chair's profit is now $" +
chair.profit());
//set the chair's weight to 7 lb
chair.setWeight(7);

Item table = new Item("Picnic Table", 70, 88);
System.out.println("The table's profit is now $" +
table.profit());
```

## Phase 2:

This is a continuation of the `Item` class we designed in the last assignment.

**Objective:** In this assignment you will enhance the `Item` class you have already created and get comfortable working with arrays of objects.

## Part I: `toString`

Add a `toString` method to your `Item` class. This method does not accept any parameters and returns a `String` reflecting the current state of the object.

Once you have written the `toString` method, the following statement will display the contents of the member variables in an Item object called chair:

```
System.out.println(chair);
```

## Part II: arrays of objects

Create an array of several `Item` objects. (You can decide on the size of the array.)

Use a *loop* to do each of the following:

- Calculate the total weight of the `Items` in the array.
- Find and output the `Item` with the highest price.
- Count how many `Items` are taxable.

You will want to consult private data members in the `Item` class so your loops will need to call accessor methods!

## Bonus:

Create a separate method for each of the tasks in **Part II**, passing the array of `Items` as a parameter.

## Phase 3:

This assignment builds on the `Item` class we designed in the last assignment. In this programming assignment, the goal is to create a *smart shopping cart*. Think of the self-checkout option in many stores these days.

Design a `ShoppingCart` class that contains `Items`. (Use the `Item` class that you already designed!) This uses *composition* since there is an array of `Item` objects inside a `ShoppingCart`. Several interesting methods are:

- **addItem** to insert an item to the cart. This is a `void` method since it modifies the state of the array in the `ShoppingCart` object. There are different ways to implement this method. One was is to have a single parameter that is an already constructed `Item`. Another way is to have a set of parameters that are the parameters to the constructor of the `Item` class.
- **cartTotal** computes the total price of the cart. This method returns a `double` but does not need any parameters since it works with data members of the `ShoppingCart` object.
- **cartTaxAmount** receives the tax rate and computes the total tax to charge for the items currently in the cart. Only Taxable items should be considered for this calculation. This method also returns a `double` but does not need any parameters since it works with data members of the `ShoppingCart` object.

For simplicity, you can assume that the capacity of a `ShoppingCart` is fixed at 10 items.

Keep in mind:

- Each class should have at least one constructor that receives arguments.
- Make data members private and create accessor and mutator methods whenever necessary.
- Each class should have a toString method that you use to display the contents of the class.

Design a driver program that allows a user to work with `ShoppingCart` objects.