**Project Overview**

A supermarket operation is involved with a number of objects such as Customer, Product, Order and Store. Product has the following information in the database: product id, price, discount percent, store id and product category. You are required to develop a program which will be able to place and process an order from a customer. The program also able compute the following operations for a store.

* Your program should be able read the information of products line by line from an input file.
* The program should be able to take the order and process the order.
* After processing the order the program also generates an invoice and display the order details of a customer.

**Reading in files of product info**

The format of the input file is the following:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Product ID** | **Price** | **Discount(%)** | **Store ID** | **Product Category** |
|  P10982 | 21 | 5.3 | S0198 | veg |

**The input file will look like the following and every line has a product details :**

P10982 21.0 5.3 S0198 veg
P10082 4.71 1.5 S0654 fruit
P10032 3.54 0.00 S0654 dairy
P10761 29.5 2.5 S0198 fruit
P10013 30.12 7.5 S0654 dairy
P10972 1.89 0.00 S0198 veg

### ****Project Materials****

Download the folder *Project1.zip*. This folder contains the following.

1. Skeleton for the three Java classes and FileReader class.

Program skeleton is given with three pre-defined classes. You will just need to compete "TO DO" section inside the class. You are not allowed to change any field variables; the given field variables are sufficient to complete the operations. You are also not allowed to change method or class definition.

**FileReader**: The class read input files send it to relevant class. This class is completed to read the lines from an input file. You don't have to do anything in this class. You will use this class by other three classes to read the input file.

**Product**: Product class take product details from the file and prepares the product information using accessor methods. No mutator methods in this class.

**Store**: The Store class communicate with Product class to gather product information to complete the store operations. This class should be able to create the number of anonymous Product type objects same as number of products we have in the input files.

**Order**: This class also communicate with product class to process the orders for a customer. The constructor in this class takes an order id to generate the invoice reference. Same as the store class you also need to create the number of anonymous Product type objects same as number of products, we have in the input files.

An input file, "input.txt", has also been provided. This contains some data that you can use for testing.

2. Three JUnit test classes, which are provided for you to check your code. Note that the test cases are not complete; a class that gets all green ticks is not guaranteed to be completely correct. Additional test cases will be used for marking and it is your responsibility to thoroughly test your code.

Feel free to use any methods from the Java libraries in your code. The following methods may be particularly useful.

* In String: length, equals, compareTo, replaceAll, replaceFirst.
* In Integer: parseInt.
* In ArrayList: many methods will be useful.

### ****Project Management Tips****

It is recommended that you tackle the project tasks in the order indicated; that you compile frequently; and that you test and run the code after completing each method, to ensure your code behaves as you think it ought, and does not fail any tests. If you are stuck on a method, it is often a good idea to look in the lecture material or in the text for an example method which is similar in some way to the one that is confusing you.

You can gain good marks even if you do not complete all the methods, so long as the code you have written compiles and runs. But if you submit a large body of code that does not compile or that crashes often, then few marks can be awarded.

Hints and tips about the various methods may be uploaded here from time to time. Whenever that happens, the document version number at the top of the page will be updated.

**Updates and tips are here**(last updated: 1.30PM, 2 April):

1) In OrderTest class, a little correcttion in testTotalItem method: should be assertEquals(2, order.getTotalItem()); instead of assertEquals(1, order.getTotalItem());

2) In Order class, getTotalPrice() and getTotalDiscount() methods don't really need to communicate with Product class.

3) In the Order class, in generateInvoice() method, the randomly generated value should be between 1 to 100.

### ****Project Assessment****

Your submission will be assessed on

* completeness: how many of the methods you have written;
* correctness: whether your methods implement the specifications exactly;
* clarity: whether your code is well-constructed and clear to read.

JUnit testing classes are provided to help you check your program before submission, but the correctness of your program remains your responsibility.

##### **Completeness and Correctness guidelines (/42)**

The marks available for each constructor and method are as follows. These numbers give some indication of how difficult we believe each component is.

|  |  |
| --- | --- |
| Product: |  constructor (2 marks), getProductID (1 marks), getPrice (1 marks), getDiscount (1 marks),  getCategory (1 marks),  getStoreID (1 marks) |
| Store: | constructor (2 marks), expensiveItem (4 marks), expensiveItemStore (5 marks), findProduct (4 marks), checkPrice (2 marks), getproductbyStoreID (3 marks), averageCost (2 marks) |
| Order: | constructor (2 marks), generateInvoice (2 marks), placeOrder (4 marks), getTotalPrice (1 marks), getTotalDiscount (1 marks), getTotalItem (1 marks), printOrder (2 marks) |

Methods will be assessed for correctness independently, so e.g. even if your standardise is faulty, other methods that use standardise could still get full marks.

Completeness and correctness will be evaluated (partly) automatically using JUnit tests. You should, therefore, try to ensure the following.

* The code passes the tests as per the specification.
* Validation should be **exactly** what is required by the assignment specification. Incorrectly rejecting a parameter will be penalized; do not add extra validation that is not asked for.
* Do not print things to System.out or System.err that are not asked for. Use the BlueJ debugger rather than print/ln statements for debugging your code.
* Do not change the given file names or class names, or modify the signatures of existing methods.

##### **Clarity guidelines (/10)**

* Keep code as simple as possible for the job it is required to do.
* Use appropriate programming constructs for all implementations.
* Do not add fields to the classes.
* Give appropriate names to all variables.
* Lay out code neatly and with appropriate indentation, with lines no longer than eighty characters.
* If a method is particularly complex, add a brief comment explaining your strategy; but otherwise, do not comment code unnecessarily.