Instructions

1. This exam is closed book and "closed Internet". You may use the Internet to download files for this exam and to commit your solution, but may not use the Internet for any other purpose.

2. You may not look at any other Java code on the computer. If you open any other Java projects in Eclipse during the exam, it is considered as cheating, which means an automatic "F" for the course.

3. Check the README.md file for the project. It may contain additional information or instructions.

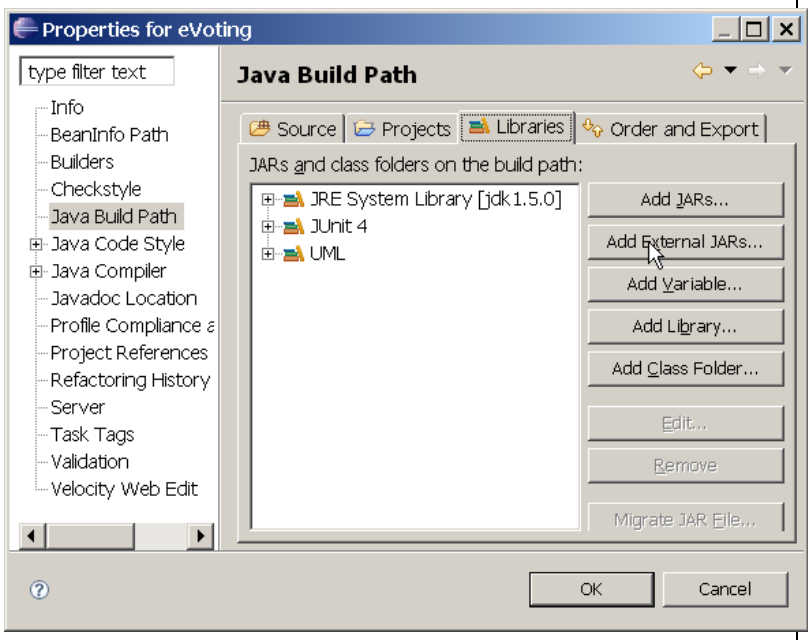
What To Submit:

1. Submit your source code to Github using the project that you downloaded the starter code from.

2. Be sure to write good Javadoc and well-formatted, readable code.

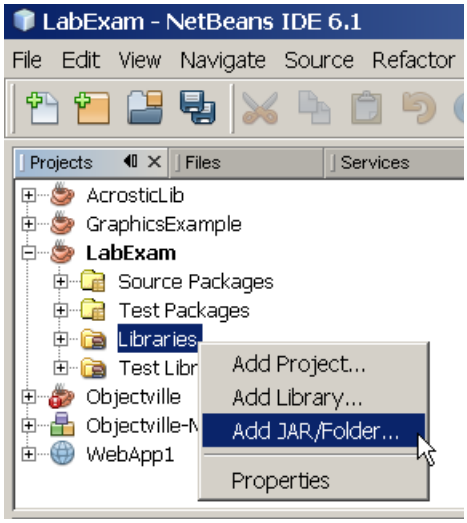
How to Add a JAR file in a Project

Eclipse



1. Open the Java Project.
2. In the "Project" menu select "Properties".
   (a) Select "Java Build Path".
   (b) Click on the "Libraries" tab.
   (c) Click "Add External JARs".
   (d) Select the jar file (store.jar)
   (e) Click OK.

NetBeans



1. Open the Java Project.
2. In the Projects window for your project, **right click** on Libraries.
3. Select "Add JAR/Folder..."
4. Select the jar file (store.jar)
5. Click on OK.

BlueJ

1. Create a Project using Project -> New Project.
2. In the Tools menu, select Preferences....
3. In the Preferences dialog box, select the Libraries tab.
4. Click the Add button.
5. Locate the jar file on your hard disk and click Open.
6. Click OK to close the Preferences dialog.

Problem 1: Print a Sale

Write a class named SalesReport to print details of a Sale and compute the total value of the Sale. Use the package store for your code.

Your code will use the **`Store`**, **`Sale`**, and **`LineItem`** classes that are in the JAR file for this exam. Those classes are in a package named "store". No source code is given for those classes.

public class Store

manages the Sale objects. Each Sale has an id number.

static Store getInstance()	return a singleton instance of the Store.
Sale getSale(long id)	get the Sale object with requested id. Returns a Sale or null if no Sale with a matching id. id must be positive number. (1-10 for this exam)
Iterator<Sale> iterator()	get all the Sales. This method is needed for Problem 3.

public class Sale implements Iterable<LineItem>

Sale contains a list of the LineItem's that a customer bought as part of a transaction (a sale). A Sale also has an id number to identify it.

public Sale(long id)	Constructor for a new Sale. You don't need this method.
Iterator<LineItem> iterator()	returns an Iterator for all the LineItem objects in this sale.
List<LineItem> getItems()	returns a List of all the LineItems in this sale. This is an alterative way to access the items in the sale. The List may be *immutable*.
long getId( )	Return the sale id of this Sale.

Sale implements Iterable<LineItem>. This means you can use it as the source in a "for - each" loop:

```
for( LineItem item: sale ) ...
```

public class LineItem

something the customer bought. A LineItem tells you what product was bought, how many units of the product, the price for each unit, and the total price of the LineItem.
An example LineItem is: 3 units of "*Oisha Green Tea*" with unit price 18. each, and total price of 54 Bt.

String getDescription()	Get a description of this item, e.g. "Oishi Green Tea".
long getProductId( )	Get the product ID for this item.
int getQuantity( )	Get the number of units of this product that customer bought
double getTotal( )	Get the *total* price for this LineItem. This is usually quantity*unitPrice.
double getUnitPrice( )	Get the price for one unit of this product.
LineItem(long productId)	Create a new LineItem with given product id. Product description and unit price are initialized using data from the Store. quantity is set to 0.
LineItem(long productId, String description, double unitPrice, int quantity)	Create a new LineItem using values specified in the call to the constructor. Its the programmer's responsibility to ensure that the description and unitPrice are correct for the product id.

void setQuantity(int qnty)	Set the number of units bought by this LineItem.

What To Write

Write a class named SalesReport that contains these methods. **These are the only methods you should write.**

double getTotal(Sale sale)	return the total price of all items in a Sale
void printItems(Sale sale)	print a formatted list of items in a sale, as in the example below
static void main(String[] arg)	main method should ask the user to input a Sale id. It gets the Sale with requested id from the Store, creates a SalesReport object, and calls the printItems() method to print everything in the sale.

Example of Running the Application

1. The **main** method will ask the user to input a Sale ID number.
2. Get the Sale from the Store using the sale id number.
3. Call printItems to print the sale. For each LineItem in the Sale, print the quantity, description, unit price, and total price. For nice output use printf such as:

```
printf("%4d %-30.30s %10.2f %10.2f", qnty, description, ...)
```

printItems must call getTotal(sale) to get the total. Don't compute the total itself. (DRY)

Here is an example:

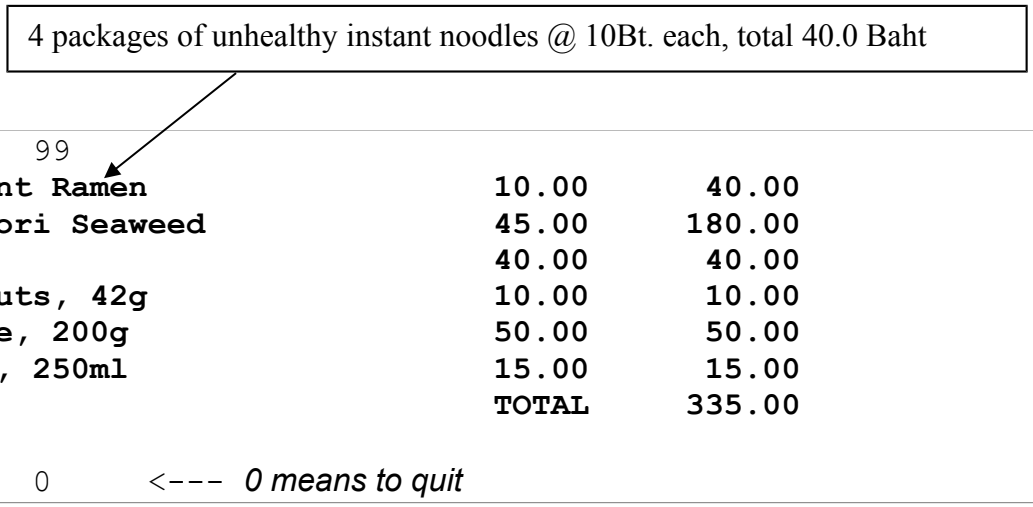
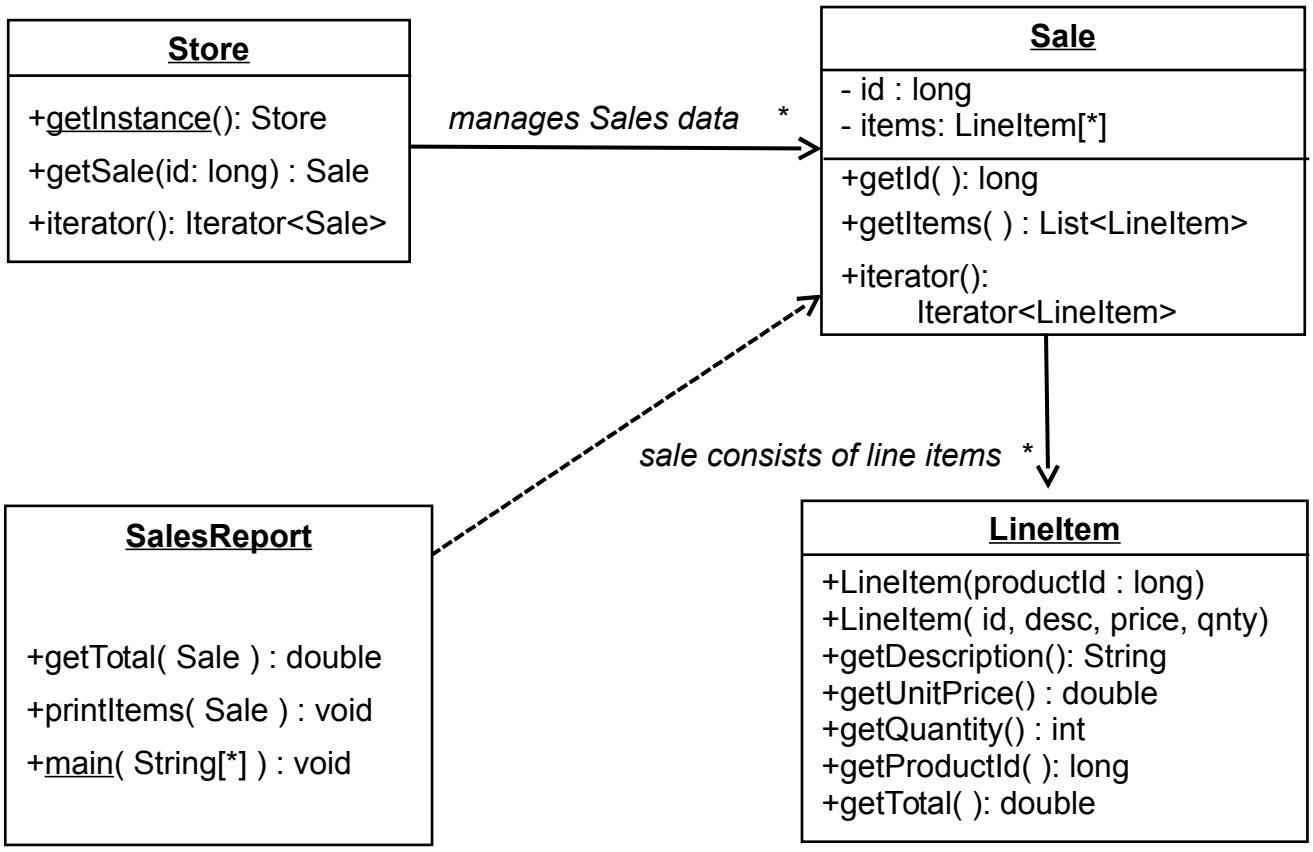


Diagram of Sale Classes

The Store records all the sales made (sales are created at a POS register such as in 7-11, and then saved in the Store). Each Sale has an id and a list of the items a customer bought. One item (called a LineItem) refers to one product the customer bought, along with the quantity he bought, and the price. LineItem knows the unit price (price for 1 unit of product) and the total price.



*LineItem records what product a customer bought, the quantity, and the price each. It computes the total price of the items (usually just quantity * price).*

Problem 2: Write a Comparator for LineItem

Write an implementation of java.util.Comparator for LineItem that orders the LineItems by the product description. Name the class **`LineItemCompareByName`**.

To test your class, write a **`main`** method that gets Sale #1 from the Store, get all the items from the Sale, sort them by name, and then print them.

Be careful! sale.getItems() returns an *immutable list* of items. You will need to copy the list in order to sort it. ArrayList has a constructor that makes a copy of another Collection.

To print the items, consider *reusing* the **`printSale( )`** method from Problem 1. It is OK to change the parameter on **`printSale`** to be: **`printSale(Iterable<LineItem> sale)`**. A nicer solution is to write this as a separate method with parameter (Iterable<LineItem> sale) and have the original printSale method call this one to print the list of LineItems.

This works because both Sale and List<LineItem> implement the Iterable interface.

Problem 3: Daily Sales Report for the Store

Write a class named **`SalesAnalyzer`** that prints a report of how many of each item that the Store sold. This is for tracking inventory, so we only want the quantities of products sold, not the value of the sales. Design the methods for this class yourself. But your code must do this:

1. Get all the Sales from the Store using the Store's iterator method.

2. Count the total quantity of each product sold in all the sales (use productId to identfy product).

3. Don't assume the productId are sequential (they are not). Hint: use a Map to store counts by productId. Create a LineItem to store the count, because a LineItem also has the product id & description.

4. Write a main() method that causes the sales to be summarized and prints a report.

Points are given for good design. A good design has simple methods that separate the tasks of getting sales from the store, summarizing the sales (counting items sold), and printing a report.

Print a report with the products *sorted alphabetically by product description*. Here is an example report:

```
  ID  Description                  Quantity Sold
  14  Banana Cake, 250g                   18
  18  Banana, kg                           8
  24  Carrot Cake, 200g                    8
 104  Drinking Water                       6
  19  Guava, kg                            8
 103  Ice Coffee, 250ml                    5
  13  Japanese Nori Seaweed                9
  11  Jasmine Brown Rice, 2kg              7
  12  Mama Instant Ramen                   7
  17  Mandarin Orange, kg                 12
 102  Oishi Green Tea                     10
  16  Red Apple, pc                        9
  20  Roast Peanuts, 42g                  14
 101  Vita Soymilk                        10
```