# IT 2045C Computer Programming II Prof. Tom Wulf Lab 07A: Software Engineering Camp Part A: Invoice Spring 2022 10 pts

This is the first is a series of sub-assignments in which we will practice the software engineering process presented here.

For each project we will:

* Derive a candidate class list from, requirements documents, Use-Cases. And subject matter knowledge
* Create CRC cards for the candidate classes and cull the unneeded classes
* Create a UML diagram showing the relationships between the classes in our design
* Implement our classes and create JUnit tests for them. (Note that we may not create the complete applications here for each case study and then we only stub out the methods in the classes to create a test first Agile software engineering approach.

## Part A: Invoice

For this first run, I will work through the design process with you using an example from our old textbook Big Java Early Object 2e. The example creates an invoice display.  
  
Task description: *An application that allows a user to enter data for an invoice, consisting of line items where the line item has quantity, a calculated total, and a product with a name and unit price. The line item quantity is the unit price multiplied by the quantity. The invoice has a total amount due which is the sum of the line item quantities.*  
  
Here is the sample: Be sure to include all the info fields and labels and the layout.

Invoice has a Title  
 a Customer Address block  
 the total amount due   
 and Line items  
  
Line Items have a product and Quantity and calculated total  
  
Product has a Name and Unit Price  
  
No need to render the graphic elements use text line dividers.

1. Using the requirements documents and general background business knowledge, we create a list of candidate classes for our application.   
     
   **Put your list here with candidate class names one per line.**

* Invoice
* Line Item
* Product

1. Now, create the CRC cards for each class noting the class name, class responsibilities, and any collaborators. At this point, you can cull the classes from the candidate list that you do not use. LEAVE THEM IN THE LIST BUT USE THE STRIKETHROUGH FONT TO SHOW THEY ARE ELIMINATED.  
     
   **Put the CRC cards here for the classes:**

Below is a representation of the CRC card.

1. **Invoice:**

**Responsibilities:**

* Manage the collection of line items.
* Calculate the total amount due based on line item quantities.
* Provide methods for adding, removing, and retrieving line items.

**Collaborators:**

* Line Item

1. **LineItem:**

**Responsibilities:**

* Store information about a single line item in the invoice.
* Calculate the total amount for the line item based on quantity and unit price.
* Provide methods for accessing and modifying quantity, unit price, and total amount.

**Collaborators:**

* Invoice

1. **Product:**

**Responsibilities:**

* Store information about a product, including its name and unit price.
* Provide methods for accessing and modifying product details.

**Collaborators:**

* Line Item

1. Now create a complete UML diagram that shows all the classes and their relationships. Don’t include the culled classes since they will not be used in the app. This should be a blueprint for coding the app.   
     
   **Insert the UML Diagram here.**

**---------------------------------------**

**| Invoice |**

**---------------------------------------**

**| - lineItems: List<LineItem> |**

**---------------------------------------**

**| + addLineItem(lineItem: LineItem) |**

**| + removeLineItem(lineItem: LineItem)|**

**| + calculateTotal(): double |**

**| + getLineItems(): List<LineItem> |**

**-------------------------------------------------------**

**|**

**|**

**|**

**---------------------------------------------------------**

**| LineItem |**

**---------------------------------------------------------**

**| - product: Product |**

**| - quantity: int |**

**---------------------------------------------------------**

**| + setProduct(product: Product) |**

**| + getProduct(): Product |**

**| + setQuantity(quantity: int) |**

**| + getQuantity(): int |**

**| + getTotal(): double |**

**--------------------------------------------------------**

**|**

**|**

**|**

**v**

**--------------------------------------------------------**

**| Product |**

**--------------------------------------------------------**

**| - name: String |**

**| - unitPrice: double |**

**--------------------------------------------------------**

**| + setName(name: String) |**

**| + getName(): String |**

**| + setUnitPrice(unitPrice: double) |**

**| + getUnitPrice(): double |**

**--------------------------------------------------------**

1. Now if you created UML Use-cases include those here:

(Not Needed for Part A)

1. Implement a Java Swing GUI Project that allows a user to enter the data for an invoice and then displays it either to the console or in a JTextArea.   
   **Project Name: Lab07A\_Invoice**

Be sure to use the classes that you developed here.   
  
A screenshot of a computer

Description automatically generated

Use the Canvas Assignment mechanism to submit the link to your GitHub repo for this project and after inserting your screen and code shots this file renamed **LastnameFirstname \_Lab\_07A.docx** (using your actual name).