TRINITY COLLEGE
DEPARTMENT OF COMPUTER SCIENCE
CPSC 215: Data Structures and Algorithms
Instructor: Dr. Chandranil Chakraborttii
Spring 2025

Laboratory 3

Objective:

The objective of this assignment is to integrate various Java concepts, including classes and objects, exceptions, inheritance, generics, abstract classes, and interfaces, to develop a Smart Inventory System. This system will manage different types of products with varying properties and behaviors.

Goal:

The goal of this assignment is to reinforce your understanding of fundamental concepts in Java, including classes and objects, exceptions, inheritance, generics, abstract classes, and interfaces.

Academic Honesty Policy

In working on programming assignments, you may discuss broad issues of interpretation and understanding and general approaches to a solution. However, conversion to a specific solution or to program code must be your own work. Programming assignments are expected to be the work of the individual student, designed, and coded by him or her alone. Violations are easy to identify and will be dealt with promptly according to the Academic Integrity and Intellectual Dishonesty outlined in the Student Handbook.

- Copying another person's programs or encouraging or assisting another person to commit plagiarism is cheating. In particular, the following activities are strictly prohibited:
- Giving and receiving help in the actual development of code or writing of an assignment.
- Looking at another person's code or showing your code to another person.
- Sharing a copy of all or part of your code regardless of whether that copy is on paper or in a computer file.
- Turning in the work of any other person(s) (former students, friends, textbook authors, people on the Internet, etc.) and representing it as your own work.
- Fabricating compilation or execution results.
- Use of AI ChatBots for any help regarding the assignment is **strictly prohibited.**

The penalty for cheating is a failing grade (F) for the course, and the student is asked to appear at an Academic Dishonesty Hearing. In addition, the College also places a record of the incident in the student's permanent record. It is your responsibility to protect your work from unauthorized access. Do not discard copies of your programs in public places. Do not leave computers unattended and copies of output lying around.

——

Assignment: Smart Inventory System

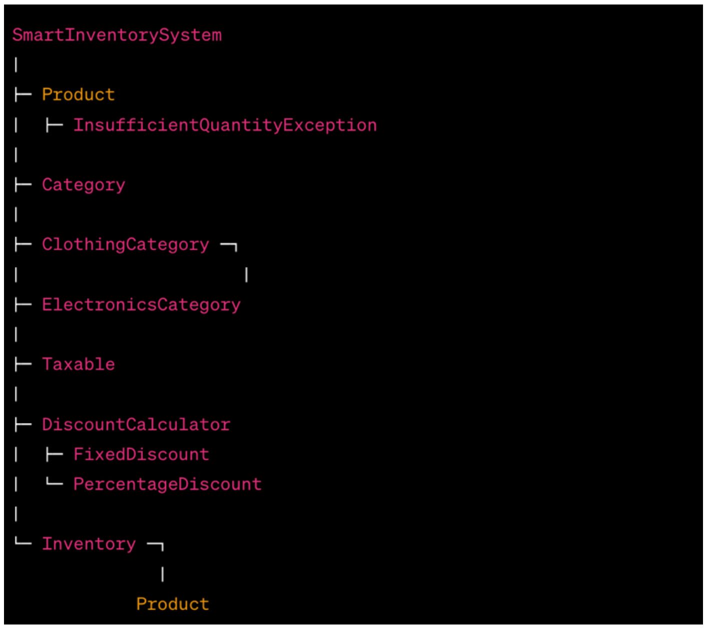
Getting Started:

- Open Eclipse and create a Java Project (File -> New -> Java Project)
- Give a project name (For example: Lab_3_CPSC_215)
- Go to Project -> Right Click -> New Package
- Give a package name (For example: Lab_3_CPSC_215)
- Now to create the classes, go to package -> Right Click -> New Java class.
- Paste contents of the given files. Keep the first line in place (package package_name;)
- Understand the code structure before starting to write code.
- Complete code to complete the classes labelled "**Need to complete**" below.

Input Files (Starter Code)

- Category.java (Fully Completed)
- InsufficientQuantityException.java (Fully Completed)
- DiscountCalculator.java (Fully Completed)
- FixedDiscount.java (Fully Completed)
- Taxable.java (Fully Completed)
- Product.java **(Need to complete)**
- Inventory.java **(Need to complete)**
- ClothingCategory.java **(Need to complete)**
- ElectronicsCategory.java **(Need to complete)**
- PercentageDiscount.java **(Need to complete)**
- SmartInventorySystem.java* **(Need to complete)** – `Contains main function.`

Here's a text-based mapping showing all class dependencies:



```
SmartInventorySystem
|
├─ Product
|   ├─ InsufficientQuantityException
|
├─ Category
|
├─ ClothingCategory ─┐
|                    |
├─ ElectronicsCategory
|
├─ Taxable
|
├─ DiscountCalculator
|   ├─ FixedDiscount
|   └─ PercentageDiscount
|
└─ Inventory ─┐
              |
         Product
```

In this mapping:

- **SmartInventorySystem** depends **on Product, ClothingCategory, ElectronicsCategory, PercentageDiscount, FixedDiscount, and Inventory**.
- **Product** depends **on InsufficientQuantityException.**
- **DiscountCalculator** is a superclass for **FixedDiscount** and **PercentageDiscount.Inventory** depends on **Product**.

Additional Details:

- **SmartInventorySystem**:
    - Depends on: Product, ClothingCategory, ElectronicsCategory, PercentageDiscount, FixedDiscount, and Inventory.

- **Product**:
    - Inherits from no class.
    - Depends on: InsufficientQuantityException.
    - Implements no interface.

- **InsufficientQuantityException**:
    - Inherits from: Exception.
    - Implements no interface.

- **Category**:
    - Inherits from no class.
    - Implements no interface.

- **ClothingCategory**:
    - Inherits from Category.
    - Implements Taxable interface.

- **ElectronicsCategory**:
    - Inherits from Category.
    - Implements Taxable interface.

- **Taxable**:
    - Defines the Taxable interface, which is implemented by ClothingCategory and ElectronicsCategory.

- **DiscountCalculator**:
    - Abstract class.
    - Depends on no other class or interface.
    - Provides an abstract method for calculating discounts.

- **FixedDiscount**:
    - Inherits from DiscountCalculator.
    - Implements no interface.

- **PercentageDiscount**:
    - Inherits from DiscountCalculator.
    - Implements no interface.

- **Inventory**:
    - Inherits from no class.
    - Depends on Product.
    - Implements no interface.

Tasks:

- **Review the Product Class:**
    - **Review** the Java class named "Product" with attributes such as product ID, name, quantity, and price.
    - **Review** the appropriate methods for displaying product details.
    - **To do:** Complete setQuantity and handle exception if quantity < 0.

- **Review how we Handle Product Exception: [Completed]**
    - **Review** the exception class, "InsufficientQuantityException," to handle scenarios where the quantity of a product falls below a specified threshold.
    - We utilize this exception within the product-related operations.

- **Implement Product Categories (Inheritance):**
    - **Review** the base class named "Category" with common properties for different product categories.
    - **Review** how we extend the "Category" class to derive two specific categories, - "ElectronicsCategory" and "ClothingCategory."
    - **Review** how these categories are integrated with the Product class using inheritance (with ClothingCategory and ElectronicsCategory classes)
    - **To do:** Complete calculateTax(double price) method in ClothingCategory and ElectronicsCategory class.

- **Generic Inventory Management (Generics):**
    - **Review** generic class named "Inventory" to manage different products.
    - **Review** methods to add/remove products and display the inventory.
    - **Review** flexibility of the generic class by handling products of various types.

    - **To do:** Complete displayInventory method to display current inventory of products.

- **Abstract Class for Discount Calculation:**
    - **Review** the abstract class named "DiscountCalculator" with an abstract method for calculating discounts based on specific rules.
    - **Review** two separate classes that extend "DiscountCalculator," such as "PercentageDiscount" and "FixedDiscount."
    - **To do:** calculateDiscount(double price) method in FixedDiscount and PercentageDiscount classes

- **Interface for Tax Calculation:**
    - **Review** the interface named "Taxable" with a method for calculating taxes.
    - **Review** how we implement the "Taxable" interface in the Product class to calculate taxes for each product.

- **Smart Inventory System Operations:**
    - Review the Smart Inventory System in action by creating instances of products from different categories.
    - Simulate product additions, removals, and adjustments in quantity.
    - Apply discount calculations and tax calculations.
    - **To do:** Complete code to add products and display inventory.
    - **To do:** Complete remaining to do statements.

- **Setup JUnit Testing:**
    - On the project folder, Right Click -> New -> Other.
    - In the wizard's section, type Junit and select JUnit Test case -> Next.
    - Provide same package name (Eg: Lab_3_CPSC_215), give class name SmartInventorySystemTest
    - Paste contents of the given files. Keep the first line in place (package "package_name";)
    - Paste contents on SmartInventorySystemTest.java (partially completed)
    - Review the project structure (All classes should be under the same package)
    - **Check JUnit Tests:** Right Click on SmartInventorySystemTest.java -> (Right Click) -> Run As -> JUnit Test
    - You should see your tests pass. (testProduct, testInsufficientQuantityException, testCategory, testClothingCategory, testElectronicsCategory)

      Runs: 5/5 Errors: 0 Failures: 0

- **JUnit Testing:**
  - Following the same format complete the remaining two test cases: (testFixedDiscount, testPercentageDiscount to test the implemented methods).
  - You should see your tests pass. (testInsert, testContains and testUnion)
    Runs: 7/7 Errors: 0 Failures: 0
  - Complete code to check fixed and percentage discount amount.
  - **Example**: Set Fixed discount to 50, and check whether object_name.calculateDiscount(any_amount) returns 50.
  - **Example**: Set percentage discount set to 10, and check whether object_name.calculateDiscount(100) returns 10. (10%)

- **Documentation and Reflection:**
  - Include JavaDoc (comments and documentation) to explain the purpose and functionality of each class, method, and exception.
  - Reflect on the challenges faced and lessons learned during the implementation.

——

Assignment Details:

1. **Initial Inventory:**
   The system starts with an initial inventory displaying the details of the laptop and T-shirt.
2. **Simulate Product Operations:**
   The details of the laptop are displayed, the quantity is set to 5, and an attempt is made to set the quantity to an insufficient value, triggering an InsufficientQuantityException.
3. **Apply Discounts:**
   Percentage and fixed discounts are applied to the laptop and T-shirt, respectively.
4. **Apply Tax:**
   The tax is calculated for the laptop based on the electronics category (10% tax), and for the T-shirt based on the clothing category (5% tax).
5. **Check and complete the Unit Tests**

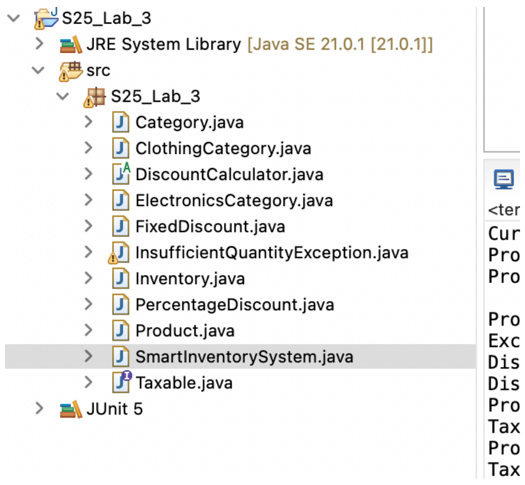
——

Submission Guidelines:

- Review the starter code. Organize your code into separate files for each class and concept as shown in starter file.
- Complete sections with “To do” statements.
- Add JavaDoc in your code.
- Upon the completion of your lab, upload your submission to the Moodle course website as a single Zip file.

——

Project Structure:

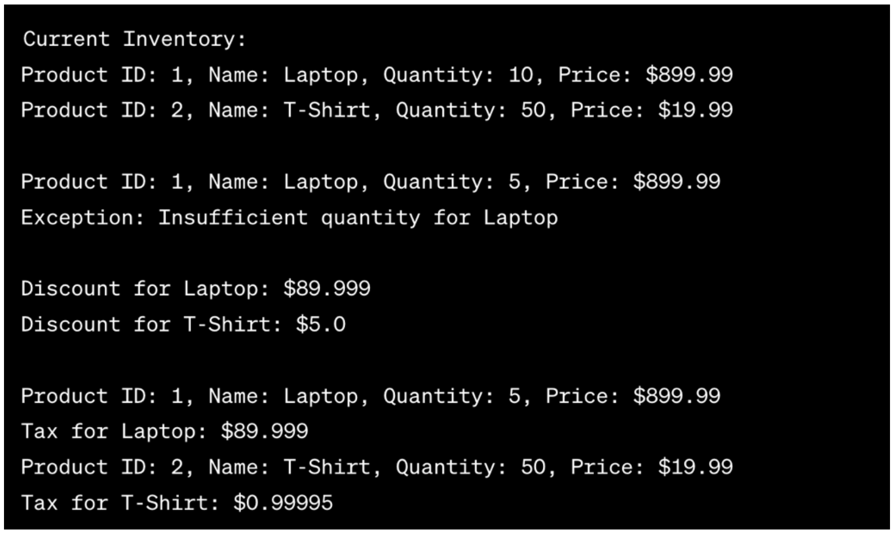


```
S25_Lab_3
  JRE System Library [Java SE 21.0.1 [21.0.1]]
  src
    S25_Lab_3
      Category.java
      ClothingCategory.java
      DiscountCalculator.java
      ElectronicsCategory.java
      FixedDiscount.java
      InsufficientQuantityException.java
      Inventory.java
      PercentageDiscount.java
      Product.java
      SmartInventorySystem.java
      Taxable.java
  JUnit 5
```

```
<ter
Cur
Pro
Pro

Pro
Exc
Dis
Dis
Pro
Tax
Pro
Tax
```

Sample output:

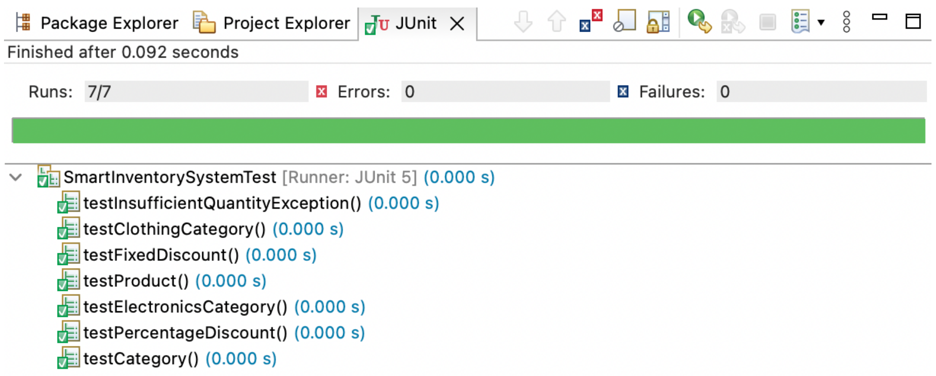


```
Current Inventory:
Product ID: 1, Name: Laptop, Quantity: 10, Price: $899.99
Product ID: 2, Name: T-Shirt, Quantity: 50, Price: $19.99

Product ID: 1, Name: Laptop, Quantity: 5, Price: $899.99
Exception: Insufficient quantity for Laptop

Discount for Laptop: $89.999
Discount for T-Shirt: $5.0

Product ID: 1, Name: Laptop, Quantity: 5, Price: $899.99
Tax for Laptop: $89.999
Product ID: 2, Name: T-Shirt, Quantity: 50, Price: $19.99
Tax for T-Shirt: $0.99995
```



```
Finished after 0.092 seconds

Runs: 7/7    Errors: 0    Failures: 0

SmartInventorySystemTest [Runner: JUnit 5] (0.000 s)
  testInsufficientQuantityException() (0.000 s)
  testClothingCategory() (0.000 s)
  testFixedDiscount() (0.000 s)
  testProduct() (0.000 s)
  testElectronicsCategory() (0.000 s)
  testPercentageDiscount() (0.000 s)
  testCategory() (0.000 s)
```