**Assignment 5**

***Assignment Requirements***

**1.** Create a class called **Invoice** that a hardware store might use to represent an invoice for an item sold at the store. The UML diagram of this class is given below:

|  |
| --- |
| **Invoice** |
| - partNo : String - description : String - quantity : int - price : double |
| + Invoice() + Invoice(partNo : String, description : String) + getPartNo() : String + setPartNo(partNo: String) : void + getDescription() : String + setDescription(description : String) : void + getQuantity() : int + setQuantity(quantity : int) : void + getPrice() : double + setPrice(price : double) : void + getInvoiceAmount() : double + toString() : String |

***Notes:***

* As shown in the UML above the class Invoice must contain four attributes, two constructors, and the corresponding accessors and mutators (get and set methods).
  + The mutator for price and quantity should throw an Exception if the quantity or price is set as zero or negative value.
* The class must also contain a method that calculates the invoice amount (i.e., multiplies the quantity by the price of item), then returns the amount as a double value.
* Finally, Invoice class must also contain a toString() method that returns a [formatted String](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html#format(java.lang.String,java.lang.Object...)) with the information of an invoice. This method will be used in the application class to display the output as shown in the examples below.
  + **Note: DO NOT** include in the toString() method the header shown in the example. That part is from the application class. Including the header in the string returned by toString() method would make the method non-generic.

**2.** Write a test app named InvoiceApp that uses class Invoice to create objects and uses different methods and exception handling techniques to perform the following:

* Prompts to and receives from the end user String values such as the part no. or part description.
* Prompts to and receives from the end user int values for the quantity.
* Prompts to and receives from the end user double values for the price.
* Use exception handling to display appropriate messages if the user enters negative or non-number values for quantity or price.
* Prints out the invoice information.

InvoiceApp must contain multiple methods as described above. Writing the code inside main method will result in mark deductions.

***Example of Interaction and Output***

**Program Prompts are in blue,**

**User Input is in red,**

**Program Output is in bold black**

**Output Sample 1:**

**Enter part no: S123**

**Enter part description: Screw**

**Enter quantity: 121**

**Enter price: 0.5**

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

**PartNo Desc Qty Price Total**

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

**S123 Screw 121 0.50 60.50**

**Output Sample 2:**

**Enter part no: S123**

**Enter part description: Screw**

**Enter quantity -79**

**Error: Quantity must be positive!**

**Output Sample 3:**

**Enter part no: S123**

**Enter part description: Screw**

**Enter quantity 67**

**Enter price: -0.45**

**Error: Price must be positive!**

**Output Sample 4:**

**Enter part no: S123**

**Enter part description: Screw**

**Enter quantity a**

**Error: Quantity must be an integer!**

**Output Sample 5:**

**Enter part no: S123**

**Enter part description: Screw**

**Enter quantity 67**

**Enter price: a**

**Error: Price must be a number!**

***Other Requirements***

* This assignment is to be done individually; you are not allowed to work on this assignment with anyone. See also the [Academic Honesty Policy](https://policy.sheridanc.on.ca/dotNet/noAuth/login.aspx?ReturnUrl=%2fdefault.aspx) for more details. Copied assignments will be subject to regulations against academic integrity.
* All assignments are subject to a late penalty of 10% per day (including weekends).
* Late assignments will only be accepted up to 3 days after the due date (including weekends).

***Submission***

**Follow these instructions carefully or you risk penalties up to 100% of your grade.**

* Your submission must follow all the submission requirements outlined in the [Submission Standards](file:///C:\Sheridan\PROG10082\references\PROG10082-ProgStandards.html).
* All assignments must be submitted in plain text format (.txt). No other types of files will be accepted for submission.
* Once you have finished your assignment and you are ready for submission, copy and paste the entire code from both java files in a text file. The file must be plain text (**.txt**) not .pdf, .doc, .rtf, .zip, etc...**Make sure that the indentations and spaces are preserved**, name the text file **YourNameAssignment5.txt** and upload it in the Assignments drop box in SLATE.

***Evaluation***

Your submission will be evaluated based on the following criteria:

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Deductions** | **Mark** |
| **Functionality** |  | **6** |
| - The input values are present; the program prompts as it is shown in the example |  | 2 |
| - Program displays the correct output and it is properly formatted |  | 2 |
| - The program displays the result as shown in the examples; the numeric values are formatted as per specifications |  | 2 |
| **Invoice Class** |  | **12** |
| - Attributes are valid identifiers and are set as private. |  | 1 |
| - Class contains at least two constructors that set initial values for some of the attributes. |  | 2 |
| - Class contains accessors and mutators for all of the attributes. |  | 2 |
| - Mutators for numeric attributes throw Exceptions to reject non positive values. |  | 2 |
| - Class contains a method for calculating the total invoice amount. |  | 2 |
| - The toString() method returns a formatted String. |  | 3 |
| **InvoiceApp Class** |  | **12** |
| - Class uses returning value methods to prompt the user for input. |  | 4 |
| - Exceptions thrown in by the mutators in Invoice class are caught and handled. |  | 3 |
| - Exceptions thrown implicitly by methods defined in JDK classes are caught and handled. |  | 3 |
| - Printing is done in a separate method that prints the results as shown in the examples. |  | 2 |
| **Penalties for not following requirements** |  |  |
| - Invoice class contains unnecessary attributes/methods (-2 marks per occurrence) |  | |
| **Penalties for Violating Programming Standards/Conventions** |  | |
| - Identifiers do not follow the naming standards and the requirements (-2 marks) |  | |
| - Indentation and spacing do not follow the standards (-2 marks) |  | |
| - Spacing between groups statements, lines, methods and class, around operators do not follow standards (-1 mark). |  | |
| - Internal program documentation is incomplete/incorrect. |  | |
| - Lines exceed 80 characters. Long lines are not broken and indented properly as per standards (-1 mark). |  | |
| **Other Penalties** |  | |
| - Required programmer ID docs missing or incomplete. (-2 marks) |  | |
| - The file name shown in the programmer ID does not match the class name. (-2 marks) |  | |
| - Submission instructions not followed (not in text format) (-100%) |  | |
| - Late submission (-10% per day up to 3 days, after that -100%) |  | |
| - Program crashes while being tested (up to -50%) |  | |
| **Assignment 5 Total:** | **30** | |

**Assignment weight 5%** of the final mark