SE116 – LAB#9 2021-2022 SPRING

Aim: Introduction to the concept of run-time polymorphism (run-time binding); understanding the use of abstract classes with abstract methods.

TO DO @ LAB:

1. Implement a Java program that simulates shopping. The user of the program will be able to add different types of products into shopping cart.

The source code consists of the following 5 classes: `Product`, `Furniture`, `PetFood`, `Book`, and `Test`.

The class `Product` is an `abstract` base class. Therefore, the class `Product` cannot instantiate any object. The class `Product` has the following `private` data members: `int ID`, `String NAME`, `double basePrice`, and `double taxRate`. The attributes `ID` and `NAME` are final instance variables. The class `Product` has the appropriate setter and getter methods for the data members. The class `Product` has parameterized constructor to initialize the data members. The class `Product` has a method `display` to print out the values of the data members. The class contains an `abstract` method named as `calculateActualPrice`. This method takes no parameter and returns `double`. As the method is `abstract` in class `Product`, it has no definition in class `Product`.

The class `Furniture` is derived directly from the class `Product`. It has the following extra `private` data members: `double width`, `double length`. The subclass `Furniture` has the appropriate setter and getter methods for the extra data members. The subclass `Furniture` has parameterized constructor/constructors to initialize the data members. If no other tax rate value is specified, then the default tax rate for any furniture will be 0.18 (18%). The subclass `Furniture` defines the inherited method `calculateActualPrice`. Therefore, the subclass `Furniture` is not an `abstract` class; it is concrete. Hence, the class `Furniture` can instantiate objects. In class `Furniture`, the method `calculateActualPrice` computes and returns the actual selling price using the following formula: `basePrice*width*length*(1+taxRate)`. The subclass `Furniture` also overrides the method `display` to print out the values of the data members and the actual selling price of the furniture.

The class `Book` is another concrete child of class `Product`. It has the following extra `private` data members: `String genre`, `int pageNumber`. The subclass `Book` has the appropriate setter and getter methods for the extra data members. The subclass `Book` has parameterized constructor/constructors to initialize the data members. If no other tax rate value is specified, then the books are tax-free items, means that the default tax rate for any book will be 0.00 (0%). The class `Book` defines the inherited method `calculateActualPrice`. In class `Book`, the method `calculateActualPrice` computes and returns the actual selling price using the following formula: `basePrice*(pageNumber*0.2)*(1+taxRate)`. The subclass `Book` also overrides the method `display` to print out the values of the data members and the actual selling price of the book.

The class `PetFood` is a direct subclass of class `Product`. It has the following extra `private` data member: `double calorie`. The subclass `PetFood` has the appropriate setter and getter methods for the extra data member. The subclass `PetFood` has parameterized constructor/constructors to initialize the data members. If no other tax rate value is specified, then the default tax rate for any pet food will be 0.08 (8%). The subclass `PetFood` defines the inherited method `calculateActualPrice`. In concrete subclass `PetFood`, the method `calculateActualPrice` computes and returns the actual selling price using the following

formula: `basePrice*calorie*(1+taxRate)`. The subclass `PetFood` also overrides the method `display` to print out the values of the data members and the actual selling price of the pet food.

The class `Test` contains the following two methods: `displayMenu`, and `main`. The method `displayMenu` prints out the following 3 options to be selected by the user: (1) adding a product into the shopping cart, (2) printing the total price of the products bought (i.e., checkout), (3) terminating the program. In method `main`, first, the shopping center needs to be filled with some products to be sold. For this reason, define an instance from `ArrayList` of `Product` and refer it as `shoppingCenter`. Then, create five products from each subclass of the superclass `Product` and fill the information of these 15 products using the data to be entered by the user. By the way, add these products all into the `shoppingCenter` in order to sell them. Next, instantiate another object from `ArrayList` of `Product` and refer it as `shoppingCart`. After this moment, a general shopping may be simulated using the following scenario: A customer searches the `shoppingCenter` for a product and then, if found, adds the corresponding product into the `shoppingCart` considering its actual price. At this point, user will see the menu with 3 options:

- First option is the addition of a product into the shopping cart. If user chooses this option, he/she will be asked for the name of the product. Then, the name value entered will be searched amongst the products in `shoppingCenter`. If the corresponding product is available in `shoppingCenter` it will be added into `shoppingCart`. If the product is not available in `shoppingCenter`, a warning will be printed out. (If there are duplicate products with same name values available in `shoppingCenter`, only the first one found will be added into `shoppingCart`.)
- Second option is the checkout. If user chooses this option, the program will traverse the items in `shoppingCart` and prints out the bill. The bill contains the corresponding information of each product as well as the actual price of each product. The bill also contains the actual total price of the shopping. After printing out the bill, the program will terminate itself automatically.
- Third option is the manual termination of the program. If user chooses this option, the program will immediately be terminated.

TO DO @ HOME:

2. Modify your project implemented in Question#**1** in the following manner:

Modify the method `main` in the following manner: If user chooses the first option (whenever user needs to add a product to his/her shopping cart), all information of each product available in `shoppingCenter` will be displayed at first. Then, the user should enter the `ID` of the product to add it into `shoppingCart`. (If there are duplicate products with same ID values available in `shoppingCenter`, only the first one found will be added into `shoppingCart`.)