**SHAAN KISHORE GUNWANI OBJECT ORIENTED PROGRAMMING WEEK 9 FORUM ANSWERS**

1. a. A class is a blueprint or template for creating objects that define a set of attributes (data) and methods (behaviours). It defines a common structure and behaviour that objects of that class will have. An instantiation of a class, also known as an object, is a specific instance of that class created using the class definition. It has its own set of attributes and can perform the methods defined by the class. For example, in the scenario given, the class "Product" can be defined to have attributes such as product name, price, and quantity, and methods such as selling and restocking. An instantiation of this class could be a specific product, such as "iPhone 12" with a price of $1000 and a quantity of 50.

b. The principles of inheritance can be incorporated into the design of this administration program by creating new classes based on existing classes. Two examples are:

The class "Manager" could be created as a subclass of the class "Employee", inheriting all the attributes and methods of the Employee class but with additional attributes and methods specific to managers, such as managing teams and setting goals.

The class "SalesPerson" could be created as a subclass of the class "Employee", inheriting all the attributes and methods of the Employee class but with additional attributes and methods specific to sales personnel, such as closing deals and tracking sales.

c. Libraries can facilitate the development of programs like this company's administration program by providing pre-written code that can be reused by the program. Libraries contain functions, classes, and other resources that can be imported into a program to perform specific tasks. For example, a library for handling financial calculations could be used to calculate salaries for managers, office staff, and sales personnel. By using libraries, programmers can save time and effort in developing their programs, as they do not have to write every piece of code from scratch. They can also benefit from the expertise of other developers who have contributed to the libraries.

1. a.

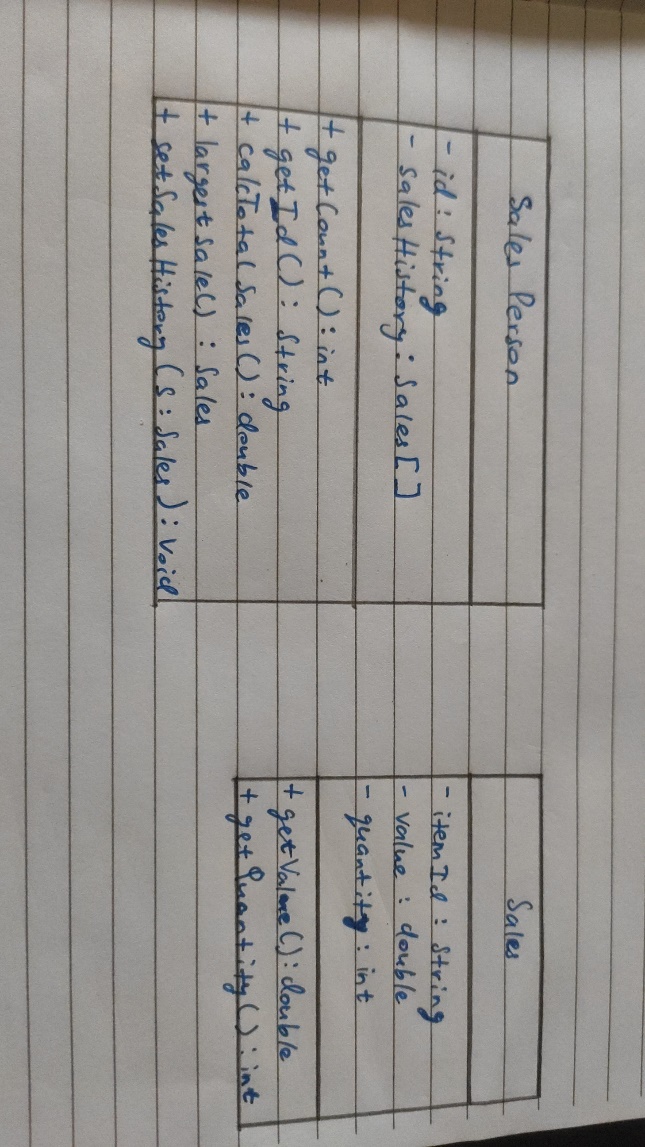
public SalesPerson(String id){

this.id = id;

salesHistory = new Sales[100]; // assuming a maximum of 100 sales per salesperson

}

b. Accessor methods are necessary for the SalesPerson class to allow access to the private instance variables from outside the class, while still maintaining encapsulation. This enables other parts of the program to retrieve information about a SalesPerson object, such as their ID or total sales, without directly accessing or modifying the instance variables.

c. i. 

c. ii. A negative effect that a future change in the design of the Sales object might have on this suite of programs is that it could require changes to the SalesPerson class as well. For example, if a new attribute were added to the Sales class, the SalesPerson class would need to be modified to accommodate this change, such as by updating the salesHistory array to include the new attribute for each Sales object. This could result in additional work for the developers and potential errors if the changes are not made correctly.

d. i. The output after running the code will be:

102

2

2550.4

1300.0

e.

public double calcTotalSales() {

double totalSales = 0.0;

for (int i = 0; i < count; i++) {

totalSales += salesHistory[i].getValue() \* salesHistory[i].getQuantity();

}

return totalSales;

}

f.

public static String highest(SalesPerson[] salesPeople) {

String highestId = "";

double highestSales = 0.0;

for (int i = 0; i < salesPeople.length; i++) {

double sales = salesPeople[i].calcTotalSales();

if (sales > highestSales) {

highestSales = sales;

highestId = salesPeople[i].getId();

}

}

return highestId;

}

g.

public static void addSales(Sales s, String id, SalesPerson[] salesPeople) {

for (int i = 0; i < salesPeople.length; i++) {

if (salesPeople[i].getId().equals(id)) {

salesPeople[i].setSalesHistory(s);

break;

}

}

}

h. To allow the Payroll class to calculate each salesperson’s salary, the Sales class needs to have a date attribute that specifies when the sale was made. The SalesPerson class also needs a method that returns all the sales made during a specific month, given a month and year.

i. Polymorphism is used in this program to allow objects to take on multiple forms, behave like objects of different types, and maintain their unique characteristics. Inheritance and interfaces are used to achieve this flexibility. For example, SalesPerson is derived from Person and Sales is derived from Item. The Payroll class can calculate the salaries of both SalesPerson and other employees using the Payable interface, despite their differences in implementation. This makes the code more adaptable to future changes in the business requirements.