Name:Christoffer Raffaelo Wijaya Student ID:2602177051

1a. The SalesPerson Class is a class, and the many salespeople, including the manager, office staff, and sales workers, are instantiations of the class.

1b. OThe office staff lesson can be expanded by the managers and sales staff classes since they are both office staff. A product class would be another illustration. The product class can then be inherited by other product kinds.

1c.You can import and using libraries, which are prewritten lines of code, in your program. It can increase the reusability of your code and increase the effectiveness of your software. By allowing the usage of existing classes and their methods rather than having to manually create such classes and methods, it also considerably reduces the amount of labor required.

(a)Complete the constructor public SalesPerson(String id), from the SalesPerson class. (2 points)

Answer:

public SalesPerson(String id)

{

this.id = id;

this.salesHistory = new Sales[100];

}

(b) Acessor methods are necessary for the SalesPerson class so that we can get controlled access to private variables of the class.

© Construct (UML) diagrams to show the relationship between the SalesPerson and Sales classes.

| SalesPerson |
| --- |
| -id: String  -salesHistory: Sales[]  -count: int; |
| +getCount(): int  +getId(): String  +setSalesHistory(Sales): void  +calcTotalSales(): double  +largestSale(): Sales |

|

|

|

↓

| Sales |
| --- |
| -itemId: String  -value: double  -quantity: int |

(c)Outline a negative effect that a future change in the design of the Sales object might have on this suite of programs.

Answer:

This is so that any necessary modifications to the SalesPerson class or other classes that rely on the Sales class can be made.Changes to the Sales object's design could lead to problems with the program's existing code.

(d) this code will not run because getSalesHistory does not exist. We can fix this by adding a getSalesHistory method in the SalesPerson class:

public Sales getSalesHistory(int x)

{

if(x >= 0 && x < count)

{

return salesHistory[x];

}

else

{

return null;

}

}

102

2

2550.4

1300.0

(e) construct the method calcTotalSales() in the salespPerson class that calculates total value of sales for a specific salesPerson

Answer:

public double calcTotalSales()

{

double total = 0.0;

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

{

total += salesHistory[i].getValue();

}

return total;

}

(f) construct a method highest(), that returns the ID of the salesperson whose sales have the largest total value.

Answer:

SalesPerson[] salesPeople;

public Driver()

{

salesPeople = new SalesPerson[100];

}

public String highest()

{

double totalSales = 0.0;

String highestSalesPerson = "";

for (SalesPerson salesPerson : salesPeople)

{

if (salesPerson != null)

{

double total = salesPerson.calcTotalSales();

if (total > totalSales)

{

totalSales = total;

highestSalesPerson = salesPerson.getId();

}

}

}

return highestSalesPerson;

}

(g) construct an addSales method in the driver class

Answer:

public void addSales(Sales s, String id)

{

for(SalesPerson salesPerson : salesPeople)

{

if (salesPerson.getId().equals(id) && salesPerson != null)

{

salesPerson.setSalesHistory(s);

break;

}

}

}

(h) The salespeople's salaries would likely likewise be determined using a calculateSalary approach.

A method to figure out how many sales were made in a certain time frame may also need to be added to the SalesPerson class.

(i)The first one uses (String id) as a parameter, whereas the second one uses (String id, Sales[] s, int c). Since all Java classes descend from the Object class, which the Sales class is a subclass of, these methods can be invoked on Sales objects.