Name:Troy Prajoga Student ID:2702291910

1a. A class is the SalesPerson Class and an instantiation of the class would be the different sales people such as the manager, office staff, and sales personnel.

1b. One example of inheritance that can be incorporated could be an employee class. The company has different types of employees and there must be attributes that they all have in common which they can potentially inherit from an employee class.

Another example of inheritance that can be incorporated could be a product class. Different products have similar attributes such as price,cost, etc.

1c.Libraries are pre-written lines of code that you can import and use in your program. Libraries can make your code more reusable and also make your program more efficient.

(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:

A change in the design of the Sales object might create compatibility issues with the existing code in the program.

(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;

}

}

If this method is added then the output of the code will be:

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) For the Payroll class, a change that should be made is an addition of a “month” variable to calculate when the end of the month is. A calculateSalary method would probably also be used in order to calculate the salary of the sales people.

(i)An example of polymorphism in this program is in the SalesPerson Class where there are two ways you can use the constructor SalesPerson(). The first is with the parameter (String id) and the second with the parameter (String id, Sales[] s, int c).