An Object-Oriented Approach to Programming Logic and Design, 4rd Edition

Chapter 2

Exercises

1. Using this book's conventions, identify each of the following as a class, method, or variable name:
   a. `calculateInsurancePremium()`
   b. `premium`
   c. `premiumValue`
   d. `paymentAmount`
   e. `InsuranceRequirements`
   f. `deductPremium()`
   g. `clientAge`

Answer:

a.	`calculateInsurancePremium()`	→	method
b.	`premium`	→	variable
c.	`premiumValue`	→	variable
d.	`paymentAmount`	→	variable
e.	`InsuranceRequirements`	→	class
f.	`deductPremium()`	→	method
g.	`clientAge`	→	variable

2. Explain why each of the following names does or does not seem like a good variable name to you.

Answer: Answers will vary. A possible solution:

a.	`p`	– legal, but too short to have much meaning
b.	`product`	– good, but could be more descriptive
c.	`productNumber`	– good
d.	`product number`	– illegal b/c of the space
e.	`pdtnbr`	– legal, but cryptc
f.	`sevenDigitProductNumberAssignedByManufacturer`	– legal and descriptive, but long
g.	`productionFor2014`	– good
h.	`2014Production`	– illegal because it starts with a digit

3. If `deposit` and `rent` are numeric variables, and `landlordName` is a string variable, which of the following statements are valid assignments? If a statement is not valid, explain why not.

   a. `deposit = 200`
   b. `rent = deposit`
   c. `rent = landlordName`
   d. `rent = "landlordName"`
   e. `850 = rent`
   f. `deposit = 150.50`
   g. `deposit = rent * 0.33`
   h. `deposit = landlordName`
   i. `landlordName = rent`
   j. `landlordName = Garvey`
   k. `landlordName = "Garvey"`
   l. `landlordName = 500`
   m. `landlordName = "500"`
   n. `landlordName = rent * 100`
   o. `landlordName = "deposit"`
   p. `500 = departmentName`
   q. `"Cooper" = departmentName`

   Answer:

   a. legal
   b. legal
   c. illegal, string cannot be assigned to a numeric variable
   d. illegal, string cannot be assigned to a numeric variable
   e. illegal, assignment cannot be made to a constant
   f. legal
   g. legal
   h. illegal, string cannot be assigned to a numeric variable
   i. illegal, assignment cannot be made to a constant
   j. illegal, string must be in quotes
   k. legal
   l. illegal, numeric value cannot be assigned to a string variable
   m. legal
   n. illegal, numeric value cannot be assigned to a string variable
   o. legal
   p. illegal, value cannot be assigned to a constant
   q. illegal, value cannot be assigned to a constant

4. Assume that `dependents = 2` and `yearsOnJob = 5`. What is the value of each of the following expressions?

   a. `dependents + yearsOnJob * 3`
   b. `10 + dependents * yearsOnJob`

c. `(yearsOnJob + 4) * dependents`
d. `4 - 3 * 2 + dependents`
e. `dependents * ((yearsOnJob - 1) * 4) - 6`

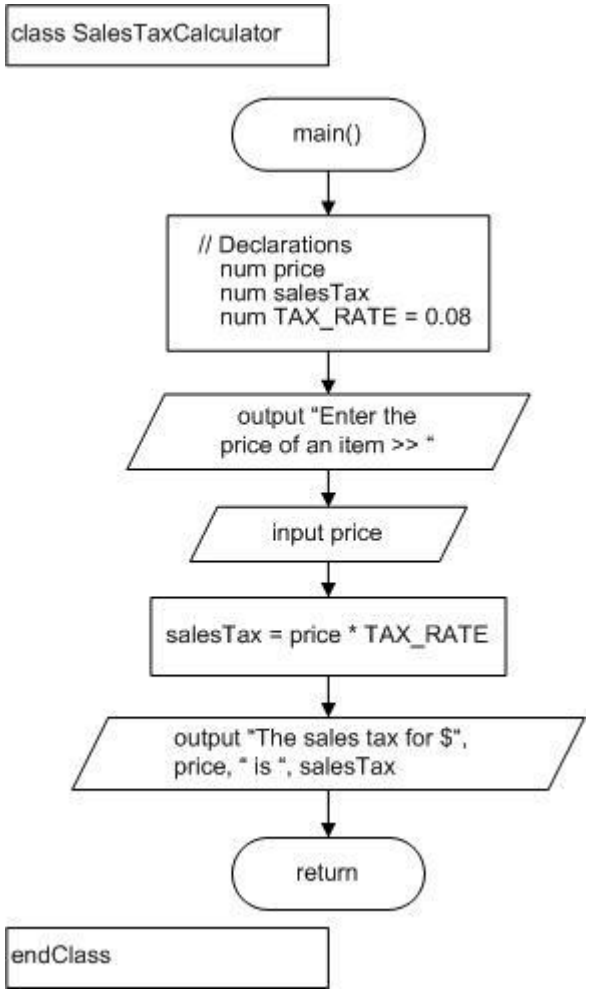
Answer:

a. `dependents + yearsOnJob * 3 = 17`
b. `10 + dependents * yearsOnJob = 20`
c. `(yearsOnJob + 4) * dependents = 18`
d. `4 - 3 * 2 + dependents = 0`
e. `dependents * ((yearsOnJob - 1) * 4) - 6 = 26`

5. Draw a flowchart or write the pseudocode for an application that allows a user to enter the price of an item and computes 8 percent sales tax on the item

Answer:

Flowchart:



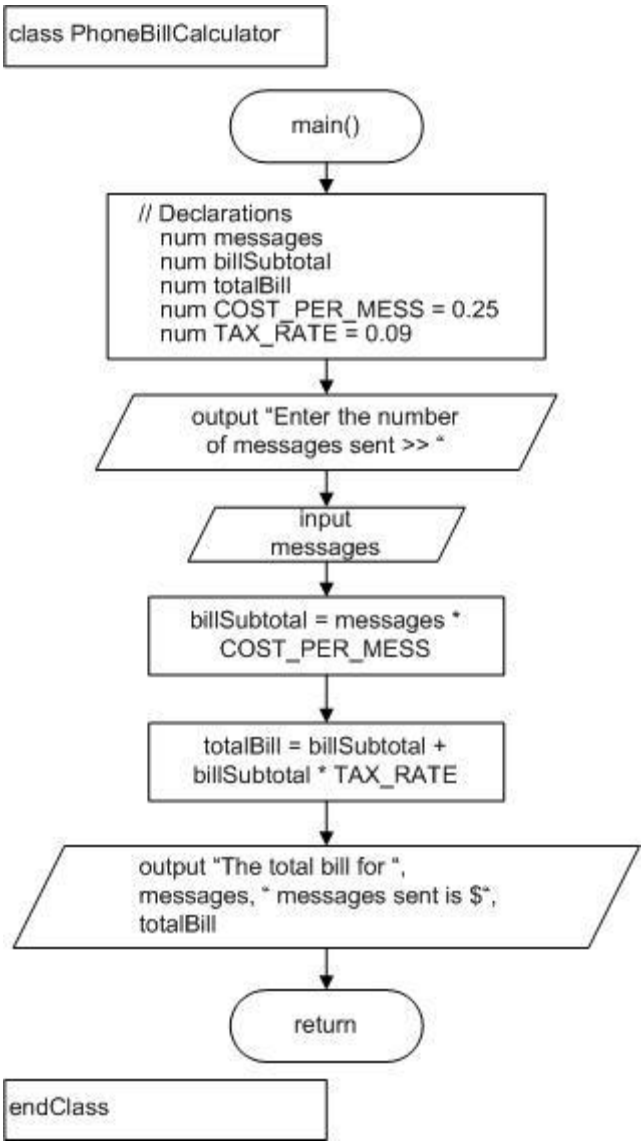
Pseudocode:

```
class SalesTaxCalculator
      main()
            // Declarations
               num price
               num salesTax
               num TAX_RATE = 0.08
            output "Enter the price of an item >> "
            input price
            salesTax = price * TAX_RATE
            output "The sales tax for $", price, " is ", salesTax
      return
endClass
```

6. Draw the flowchart or write the pseudocode for an application that allows a user to enter the number of text messages he or she sent last month and then displays the bill. Messages cost 25 cents each, and 9 percent tax is charged on the total.

   Answer:

   Flowchart:



Pseudocode:

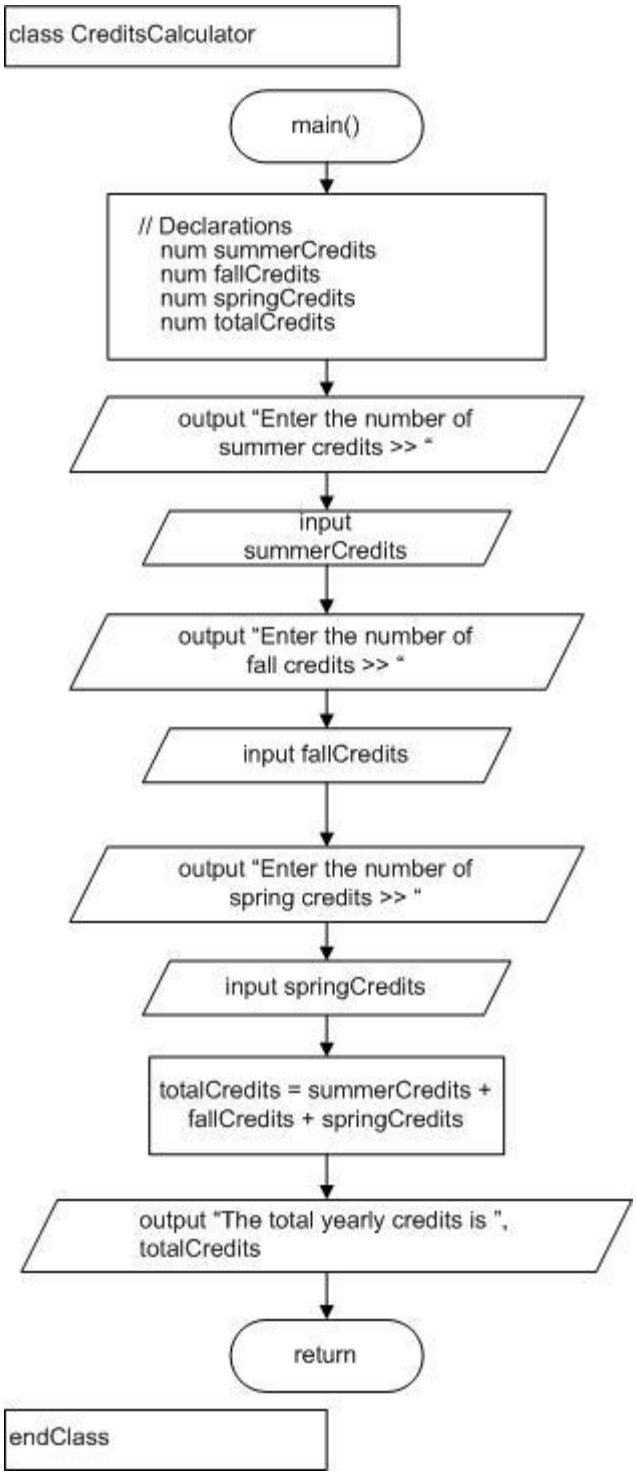
```
class PhoneBillCalculator
      main()
            // Declarations
               num messages
               num billSubtotal
               num totalBill
               num COST_PER_MESS = 0.25
               num TAX_RATE = 0.09
            output "Enter the number of messages sent >> "
            input messages
            billSubtotal = messages * COST_PER_MESS
            totalBill = billSubtotal + billSubtotal * TAX_RATE
            output "The total bill for ", messages,
                   " messages sent is ", totalBill
```

```
        return
endClass
```

7. Draw the flowchart or write the pseudocode for an application that allows a user to enter credits earned for the fall, spring, and summer semesters and then displays the total for the year.

   Answer:

   Flowchart:



Pseudocode:

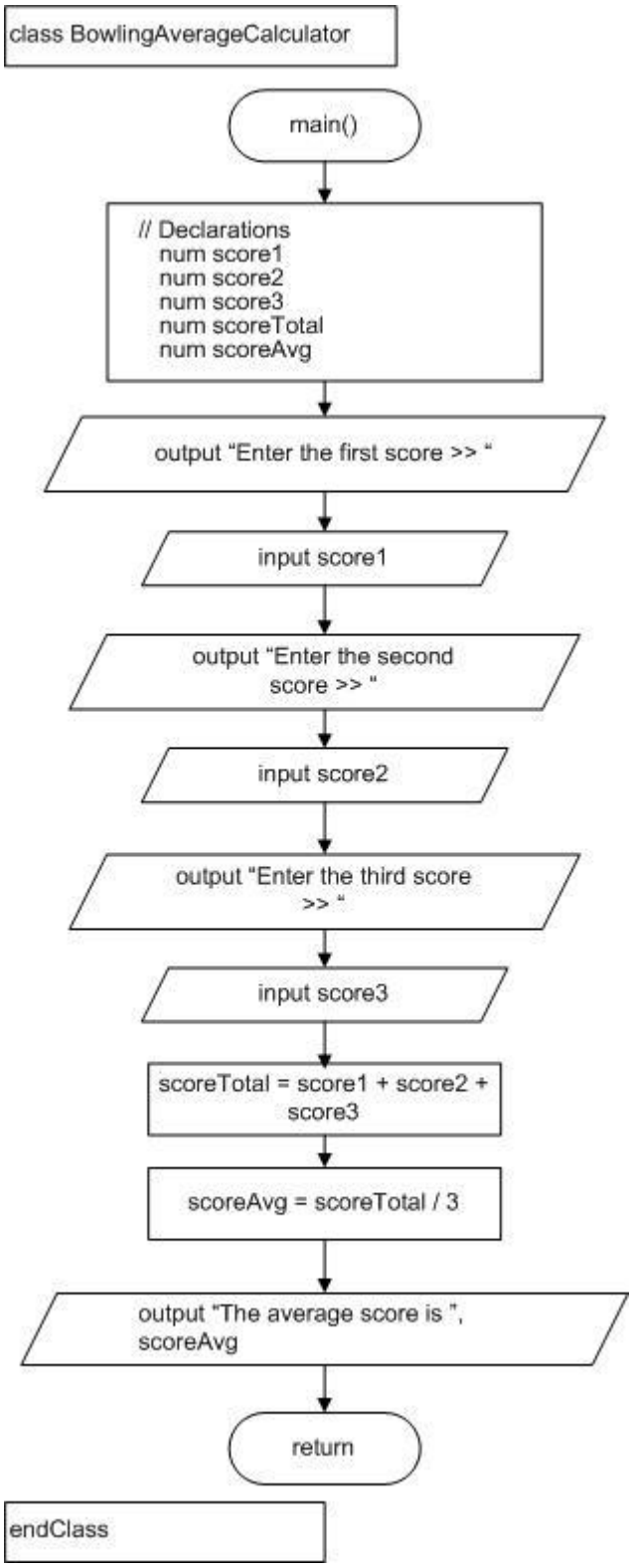
```
class CreditsCalculator
      main()
            // Declarations
               num summerCredits
               num fallCredits
```

```
                num springCredits
                num totalCredits
             output "Enter the cost of summer credits >> "
             input summerCredits
             output "Enter the cost of fall credits >> "
             input fallCredits
             output "Enter the cost of spring credits >> "
             input springCredits
             totalCredits = summerCredits + fallCredits +
                                springCredits
             output "The total yearly credits is ",
                         totalCredits
      return
endClass
```

8. Draw the flowchart or write the pseudocode for an application that allows a bowler to enter scores for three bowling games and then displays the numeric average.

   Answer:

   Flowchart:



Pseudocode:

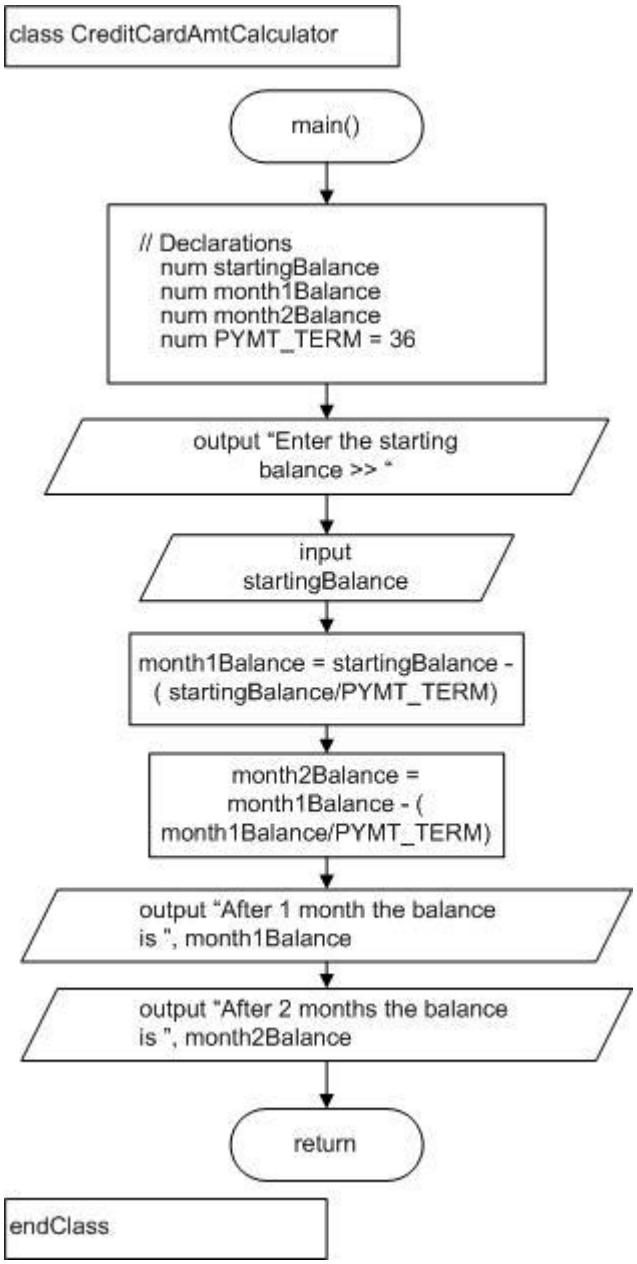
```
class BowlingAverageCalculator
      main()
            // Declarations
```

```
            num score1
            num score2
            num score3
            num scoreTotal
            num scoreAvg
         output "Enter the first score >> "
         input score1
         output "Enter the second score >> "
         input score2
         output "Enter the third score >> "
         input score3
         scoreTotal = score1 + score2 + score3
         scoreAvg = scoreTotal / 3
         output "The average score is ", scoreAvg
   return
endClass
```

9. Draw the flowchart or write the pseudocode for an application that allows a user to enter an automobile loan balance. Assume that the user pays 1/36 of the balance each month, and display the new balance after one month and after two months.

   Answer:

   Flowchart:



Pseudocode:

```
class CreditCardAmtCalculator
      main()
            // Declarations
               num startingBalance
               num month1Balance
               num month2Balance
               num PYMT_TERM = 36
            output "Enter the starting balance >> "
            input startingBalance
```

```
                month1Balance = startingBalance *
                        (startingBalance/PYMT_TERM)
                month2Balance = month1Balance *
                        (month1Balance/PYMT_TERM)
                output "After 1 month the balance is ",
                            month1Balance
                output "After 2 months the balance is ",
                            month2Balance
        return
endClass
```

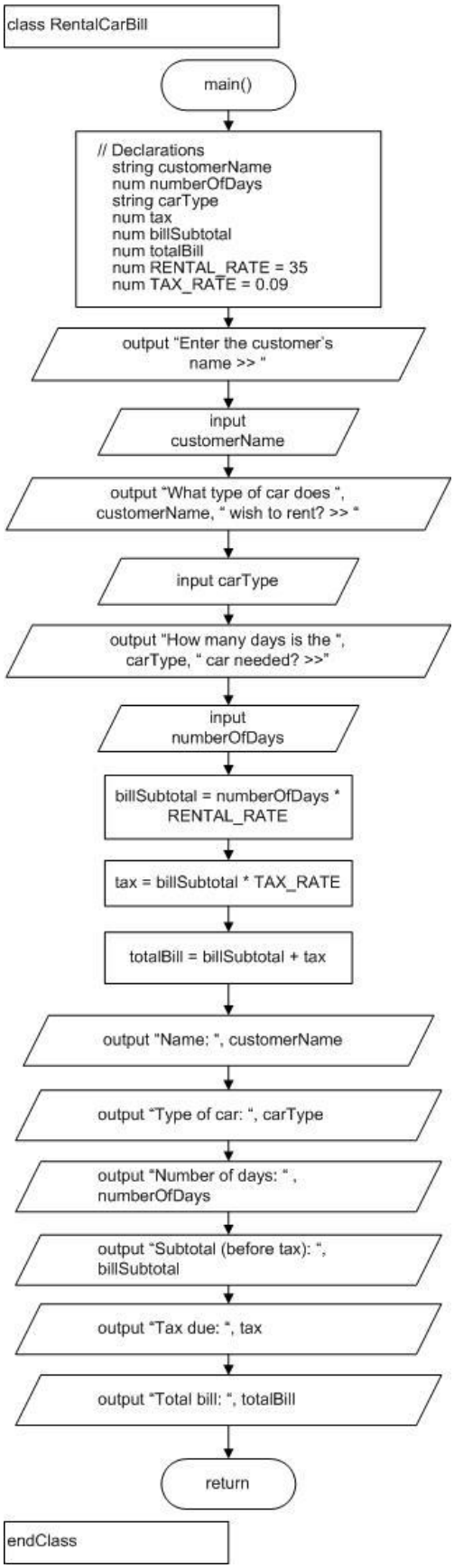
Case Problems

Case: Cost Is No Object

1. In Chapter 1, you thought about the objects needed for programs for Cost Is No Object—a car rental service that specializes in lending antique and luxury cars to clients on a short-term basis. One required application is a program that calculates customer bills. This month, cars are being rented for $35 per day, with a 9 percent tax applied. Draw a flowchart or write pseudocode for a program that accepts a client's name, the type of car the client wants to rent, and the number of rental days needed. Output the client's bill, including the name, type of car, number of days, total due before tax, tax, and total due with tax.

Answer:

Flowchart:


class RentalCarBill
main()
// Declarations
string customerName
num numberOfDays
string carType
num tax
num billSubtotal
num totalBill
num RENTAL_RATE = 35
num TAX_RATE = 0.09
output "Enter the customer's name >> "
input customerName
output "What type of car does ", customerName, " wish to rent? >> "
input carType
output "How many days is the ", carType, " car needed? >>"
input numberOfDays
billSubtotal = numberOfDays * RENTAL_RATE
tax = billSubtotal * TAX_RATE
totalBill = billSubtotal + tax
output "Name: ", customerName
output "Type of car: ", carType
output "Number of days: ", numberOfDays
output "Subtotal (before tax): ", billSubtotal
output "Tax due: ", tax
output "Total bill: ", totalBill
return
endClass

Pseudocode:

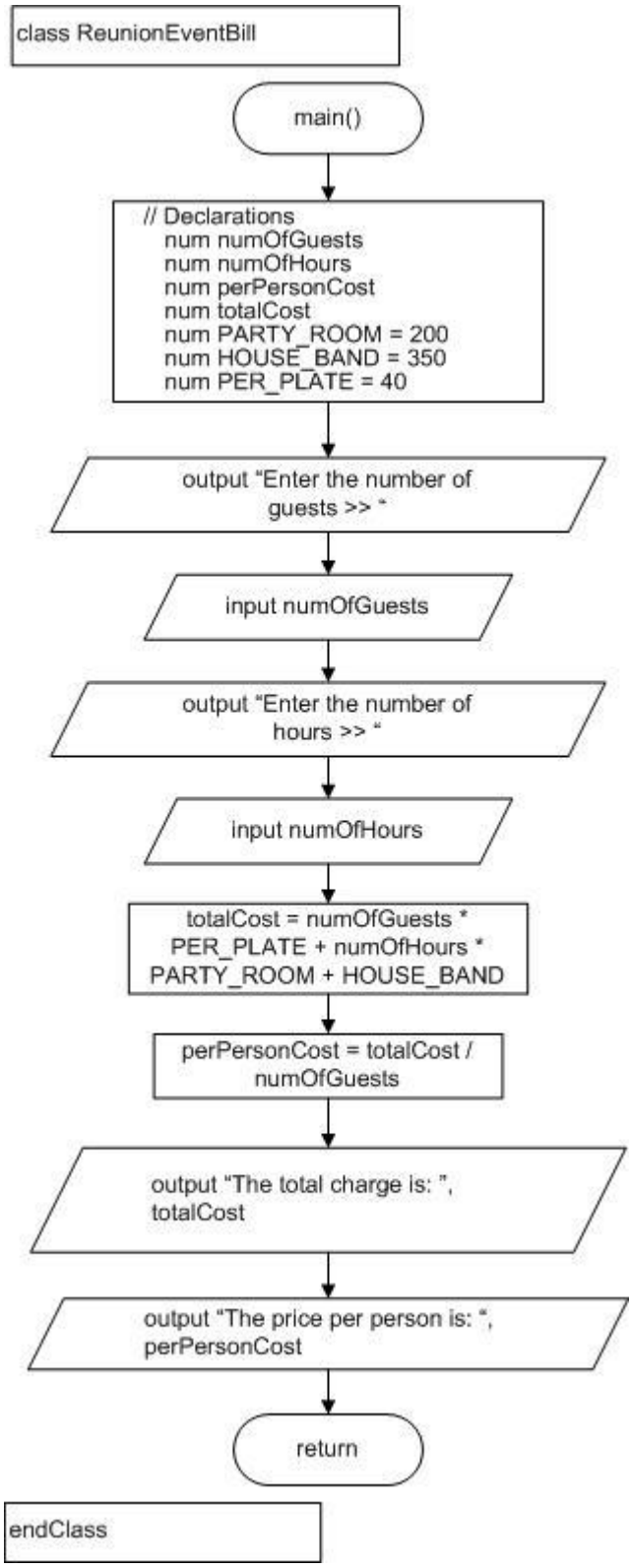
```
class RentalCarBill
      main()
            // Declarations
               string customerName
               num numberOfDays
               string carType
               num tax
               num billSubtotal
               num totalBill
               num RENTAL_RATE = 35
               num TAX_RATE = 0.09
            output "Enter the customer's name >> "
            input customerName
            output "What type of car does ", customerName,
                        " wish to rent? >>"
            input carType
            output "How many days is the ", carType,
                        " car needed? >>"
            input numberOfDays
            billSubtotal = numberOfDays * RENTAL_RATE
            tax = billSubtotal * TAX_RATE
            totalBill = billSubtotal + tax
            output "Name: ", customerName
            output "Type of car: ", carType
            output "Number of days: ", numberOfDays
            output "Subtotal (before tax): ", billSubtotal
            output "Tax due: ", tax
            output "Total bill: ", totalBill
      return
endClass
```

Case: Classic Reunions

2. In Chapter 1, you thought about the objects needed for programs for Classic Reunions—a company that provides services for organizers of high school class reunions. One required program must be able to estimate the cost of a reunion event per person. This month, the company is charging $200 per hour for renting its on-site party room, $350 for its house band for the evening, and $40 a plate for dinner. Develop the logic for an application that accepts the number of guests expected for an event and the number of hours for the party as input, then calculates and outputs the total cost for the event as well as the cost per person.

Answer:

Flowchart:


class ReunionEventBill
main()
// Declarations
num numOfGuests
num numOfHours
num perPersonCost
num totalCost
num PARTY_ROOM = 200
num HOUSE_BAND = 350
num PER_PLATE = 40
output "Enter the number of guests >> "
input numOfGuests
output "Enter the number of hours >> "
input numOfHours
totalCost = numOfGuests * PER_PLATE + numOfHours * PARTY_ROOM + HOUSE_BAND
perPersonCost = totalCost / numOfGuests
output "The total charge is: ", totalCost
output "The price per person is: ", perPersonCost
return
endClass

Pseudocode:

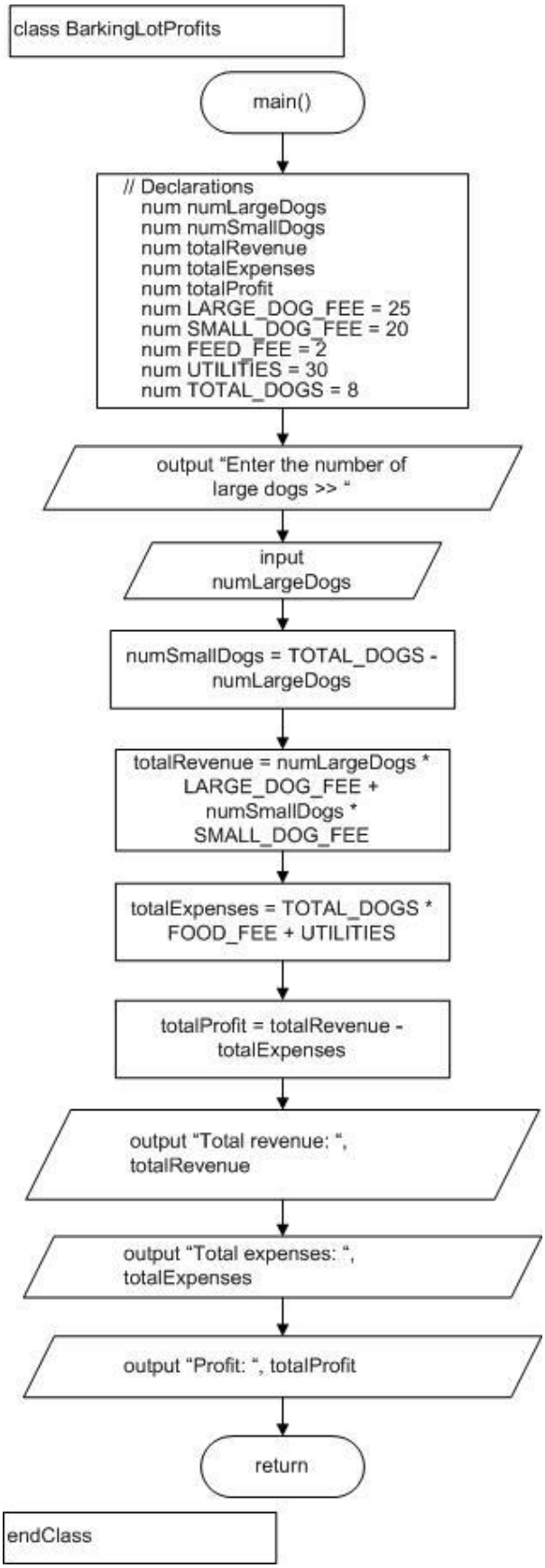
```
class ReunionEventBill
      main()
            // Declarations
               num numOfGuests
               num numOfHours
               num perPersonCost
               num totalCost
               num PARTY_ROOM = 200
               num HOUSE_BAND = 350
               num PER_PLATE = 40
            output "Enter the number of guests >> "
            input numOfGuests
            output "Enter the number of hours >>"
            input numOfHours
            totalCost = numOfGuests * PER_PLATE +
                        numOfHours * PARTY_ROOM +
                        HOUSE_BAND
            perPersonCost = totalCost / numOfGuests
            output "The total charge is: ", totalCost
            output "The price per person is: ", perPersonCost
      return
endClass
```

Case: The Barking Lot

3. In Chapter 1, you thought about the objects needed for programs for The Barking Lot—a dog boarding facility. One required program must be able to estimate profits for a day. The facility can board eight dogs at a time; it charges $25 a day for dogs that weigh more than 50 pounds and $20 a day for smaller dogs. The facility's expenses include $2 per day per dog for food (no matter the size of the dog), and $30 per day for utilities. Develop the logic for a program that allows a user to enter the number of large dogs boarded; assume that the rest are small dogs and that the facility is full. Output is the total revenue collected for the day, total expenses, and the difference.

Answer:

Flowchart:


class BarkingLotProfits
main()
// Declarations
num numLargeDogs
num numSmallDogs
num totalRevenue
num totalExpenses
num totalProfit
num LARGE_DOG_FEE = 25
num SMALL_DOG_FEE = 20
num FEED_FEE = 2
num UTILITIES = 30
num TOTAL_DOGS = 8
output "Enter the number of large dogs >> "
input numLargeDogs
numSmallDogs = TOTAL_DOGS - numLargeDogs
totalRevenue = numLargeDogs * LARGE_DOG_FEE + numSmallDogs * SMALL_DOG_FEE
totalExpenses = TOTAL_DOGS * FOOD_FEE + UTILITIES
totalProfit = totalRevenue - totalExpenses
output "Total revenue: ", totalRevenue
output "Total expenses: ", totalExpenses
output "Profit: ", totalProfit
return
endClass

Pseudocode:

```
class BarkingLotProfits
      main()
            // Declarations
               num numLargeDogs
               num numSmallDogs
               num totalRevenue
               num totalExpenses
               num totalProfit
               num LARGE_DOG_FEE = 25
               num SMALL_DOG_FEE = 20
               num FOOD_FEE = 2
               num UTILITIES = 30
               num TOTAL_DOGS = 8
            output "Enter the number of large dogs >> "
            input numLargeDogs
            numSmallDogs = TOTAL_DOGS - numLargeDogs
            totalRevenue = numLargeDogs * LARGE_DOG_FEE +
                              numSmallDogs * SMALL_DOG_FEE
            totalExpenses = TOTAL_DOGS * FOOD_FEE + UTILITIES
            totalProfit = totalRevenue - totalExpenses
            output "Total revenue: ", totalRevenue
            output "Total expenses: ", totalExpenses
            output "Profit: ", totalProfit
      return
endClass
```

Up for Discussion

1. Many programming style guides are published on the Web. These guides suggest good identifiers, explain standard indentation rules, and identify style issues in specific programming languages. Find style guides for at least two languages (for example, C++, Java, Visual Basic, or C#) and list any differences you notice.

Answer:

The style guides generally list conventions for naming variables, indenting code, and so on. Some guides suggest you capitalize variable names, others suggest you begin them all with a lowercase letter. Some C++ and Java style guides suggest using opening braces at the end of a line; others insist they be placed on a line by themselves. All guides suggest consistency within your programs.

2. What advantages are there to requiring variables to have a data type?

Answer:

When variables have data types, automatic checking for certain types of errors takes place. For example, if age is numeric, you will receive a compiler error if you attempt to assign your name to it. The computer can find meaningless, and therefore, probably invalid code. Machine instructions can be made more efficient when the compiler knows variables' types. Naming data types also serves as a form of documentation, making the programmer's intentions clearer.

3. Would you prefer to write a large program by yourself, or work on a team in which each programmer produces one or more methods? Why?

Answer:

Student answers will vary based on their preferences. Advantages of working on your own include being responsible for the entire system, being paid more, and not depending on others who might miss deadlines or produce inferior quality work. Advantages to working on a team include the camaraderie, having others off of whom you can bounce ideas, and completing a project more quickly.

4. Extreme programming is a system for rapidly developing software. One of its tenets is that all production code is written by two programmers sitting at one machine. Is this a good idea? Does working this way as a programmer appeal to you? Why or why not?

Answer:

Student opinions will vary. Many will like the idea of working with another programmer; others will detest it.

Pair programming is said to yield the following benefits:

- Increased discipline. Pairing partners are more likely to "do the right thing" and are less likely to take long breaks.
- Better code. Pairing partners are less likely to produce a bad design due to their immersion, and tend to come up with higher quality designs.
- Resilient flow. Pairing leads to a different kind of flow than programming alone, but it does lead to flow. Pairing flow happens more quickly: one programmer asks the other, "What were we working on?" Pairing flow is also more resilient to interruptions: one programmer deals with the interruption while the other keeps working.
- Multiple developers contributing to design. If pairs are rotated frequently, several people will be involved in developing a particular feature. This can help create better solutions, particularly when a pair gets stuck on a particularly tricky problem

- Improved morale. Pair programming can be more enjoyable for some engineers than programming alone.
- Collective code ownership. When everyone on a project is pair programming, and pairs rotate frequently, everybody gains a working knowledge of the entire codebase.
- Mentoring. All programmers, even beginners, possess knowledge that others don't. Pair programming is a painless way of spreading that knowledge.
- Team cohesion. People get to know each other more quickly when pair programming. Pair programming may encourage team gelling.
- Fewer interruptions. People are more reluctant to interrupt a pair than they are to interrupt someone working alone.
- One fewer workstation required. Since two people use one workstation, one fewer workstation is required, and therefore the extra workstation can be used for other purposes.
- Studies have shown that after training for the "people skills" involved, two programmers are more than twice as productive as one for a given task.

Pair programming has the following disadvantages:

- Experienced developers may find it tedious to tutor a less experienced developer in a paired environment.
- Many engineers prefer to work alone, and may find the paired environment cumbersome.
- Productivity gains or losses are hard to compare between paired and non-paired environments, as metrics of programmer productivity are controversial at best.
- Experienced engineers quite likely produce code that is very accurate, and the additional theoretical gain from pairing is not worth the cost of an additional engineer.
- Differences in coding style may result in conflict.
- In the case where the team has slightly different work schedules, which is common in an environment that values work-life balance, the pair is only available during the overlap of their schedules. Therefore, not only does it require more man-hours to complete a task, a typical day has fewer pair-hours available, which further increases the overall task completion time.
- Where a company values telecommuting (working from home) or when an employee must work from outside the office for whatever reasons, pair programming can be difficult and even impossible.