**R8.1. What is encapsulation? Why is it useful?**

Encapsulation is one of the distinctive features of object-oriented programming. Basically, it means that all the data and functionality is in one class. This is often done by making all instance variables private, thus restricting access to the class. Additionally, public methods are declared that are able to set and get the values of the instance variables. Overall, encapsulation is useful because it allows for changes to be made easily within the client class without messing up the functionality of the entire program. This also helps to prevent more common errors from happening within the client class. Finally, it only gives users access to the part of the code they are supposed to be working with.

**R8.2. What values are returned by the calls reg1.getCount(), reg1.getTotal(), reg2.getCount(), and reg2.getTotal() after these statements?**

**CashRegister reg1 = new CashRegister();**

**reg1.addItem(3.25);**

**reg1.addItem(1.95);**

**CashRegister reg2 = new CashRegister();**

**reg2.addItem(3.25);**

**reg2.clear();**

reg1.getCount() = 2, reg1.getTotal() = 5.2, reg2.getCount() = 0, reg2.getTotal() = 0

**R8.3. Consider the Menu class in Worked Example 8.1. What is displayed when the following calls are executed?**

**Menu simpleMenu = new Menu();**

**simpleMenu.addOption("Ok");**

**simpleMenu.addOption("Cancel");**

**int response = simpleMenu.getInput();**

1. Ok
2. Cancel

**R8.5. Consider the data representation of a cash register that keeps track of sales tax in Section 8.8. Instead of tracking the taxable total, track the total sales tax. Redo the walkthrough with this change.**

Original example from problem:

|  |  |  |  |
| --- | --- | --- | --- |
| reg3.totalPrice | reg3.taxableTotal | reg3.taxRate | reg3.totalTax |
| 0 | 0 | 7.5 | 0 |
| 3.95 | 0 | 7.5 | 0 |
| 23.90 | 19.95 | 7.5 | 1.50 |

New example from self-check walkthrough:

|  |  |  |  |
| --- | --- | --- | --- |
| reg1.totalPrice | reg1.taxableTotal | reg1.taxRate | reg1.totalTax |
| 0 | 0 | 5 | 0 |
| 5 | 0 | 5 | 0 |
| 11 | 6 | 5 | 0.30 |
| 15 | 10 | 5 | 0.50 |

**R8.6. Suppose the CashRegister needs to support a method void undo() that undoes the addition of the preceding item. This enables a cashier to quickly undo a mistake. What instance variables should you add to the CashRegister class to support this modification?**

You could add an instance variable for the last total price (which would store the total price before the most recent item was added) as well as an instance variable for the last taxable total (which should store the taxable total before the most recent item was stored). The undo method could then decrement itemCount and reset the total price and the taxable total. However, this method can also be done without adding instance variables as you could simply decrement itemCount and subtract the price of the item from totalPrice.

**R8.7. What is an instance method, and how does it differ from a static method?**

An instance method is a method that requires that an object of the class is created before it is called. Basically, an object of the class must be created in the class it was defined before an instance method is able to be called. Contrary to instance methods, static methods can be called without creating an object of that class first. Additionally, instance methods are able to access instance variables directly whereas static methods can’t do this. Finally, static methods must be declared with the keyword static, whereas instance variables don’t need to.

**R8.8. What is a mutator method? What is an accessor method?**

Both a mutator method and an accessor method are used with private instance variables in object classes. More specifically, a mutator method, also known as a setter, is used to set a data field to a new value. Mutator methods typically take in a parameter of the same type of the variable they’re setting. On the other hand, accessor methods, often referred to as getters, return the current value of a data field. Getters usually just return the instance variable they are meant to deal with.

**R8.11. What is a constructor?**

A constructor is a public method in an object class with no return type, and it is what is called by the class’s object in the client class. More specifically, a constructor is typically used to initialize the instance variables of an object (typically named the same thing as the class it’s in). Additionally, constructors can be used with the keyword new to call the object outside of the object class. Constructors also often have parameters that allow a user to initialize the instance variables to specific values. Finally, classes can have more than one constructor, each with different parameters.

**R8.12. How many constructors can a class have? Can you have a class with no constructors? If a class has more than one constructor, which of them gets called?**

As mentioned in the question above, a class can have many constructors. Additionally, it is also possible to have a class with no constructors. However, if you do not explicitly create a constructor, the Java compiler will create one in the form:

public className(){

//no code inside

}

Finally, if a class has more than one constructor, the constructor with the correct amount/type of parameters provided in the client class will be called.

**R8.13. Using the object tracing technique described in Section 8.8, trace the program at the end of Section 8.7.**

|  |  |  |
| --- | --- | --- |
| itemCount | price | totalPrice |
| 0 | 0 | 0 |
| 1 | 1.95 | 1.95 |
| 2 | 0.95 | 2.90 |
| 3 | 2.50 | 5.40 |

**R8.14. Using the object tracing technique described in Section 8.8, trace the program in How To 8.1.**

|  |  |
| --- | --- |
| balance | interest |
| 1000 | 0 |
| 1500 | 0 |
| 1490 | 1 |
| 1504.90 |  |

**R8.15. Design a modification of the BankAccount class in How To 8.1 in which the first five transactions per month are free and a $1 fee is charged for every additional transaction. Provide a method that deducts the fee at the end of a month. What additional instance variables do you need? Using the object tracing technique described in Section 8.8, trace a scenario that shows how the fees are computed over two months.**

You will now need an instance variable for the number of transactions as well as the total fee. The transactions variable should be incremented by 1 every time a deposit or withdrawal occurs. You could then define a method like the following:

public void transactionFee(){

if(transactions > 5){

fee = fee + 1;

}

}

You could then call this method at the end of both the deposit and withdraw methods. Additionally, a method is needed for the monthly fine:

public void monthlyFine(){

balance = balance – fee;

fee = 0;

transactions = 0;

}

This method should be called at the end of each month in order to charge the total monthly fee.

Month 1:

|  |  |  |
| --- | --- | --- |
| balance | counter | fine |
| 2000 | 0 | 0 |
| 3000 | 1 | 0 |
| 2750 | 2 | 0 |
| 2700 | 3 | 0 |
| 2890 | 4 | 0 |
| 4560 | 5 | 0 |
| 4500 | 6 | 1 |
| 4499 | 0 | 0 |

Month 2:

|  |  |  |
| --- | --- | --- |
| balance | counter | fine |
| 4499 | 0 | 0 |
| 4400 | 1 | 0 |
| 4500 | 2 | 0 |
| 3932 | 3 | 0 |
| 3992 | 4 | 0 |
| 4118 | 5 | 0 |
| 9008 | 6 | 1 |
| 8941 | 7 | 2 |
| 8847 | 8 | 3 |
| 8844 | 0 | 0 |

In this example, the balance after two months would be $8844.

**R8.16. Instance variables are “hidden” by declaring them as private, but they aren’t hidden very well at all. Anyone can read the class declaration. Explain to what extent the private reserved word hides the private implementation of a class.**

The private keyword is used to make it so that instance variables can only be accessed within the same class and not in any other classes. In this sense, instance variables are “hidden” from all classes except for the one they are declared in. Additionally, if an instance variable is declared as private, it can’t be directly accessed in a call to the object. Instead, a getter method must be used in order to access the value of private instance variables.

**R8.17. You can read the itemCount instance variable of the CashRegister class with the getCount accessor method. Should there be a setCount mutator method to change it? Explain why or why not.**

Yes, there should be a setCount mutator method in the CashRegister that allows the user to change the value of the itemCount variable. This is because itemCount is a private variable, so it can’t be set to a new value in the main class. As a result, a public setter method is needed to be able to change the value of itemCount from the client class.

**R8.18. In a static method, it is easy to differentiate between calls to instance methods and calls to static methods. How do you tell them apart? Why is it not as easy for methods that are called from an instance method?**

In a static method, a new object must be declared before you can call an instance method. The name of the new object is then used to call the instance method. On the other hand, to call a static method you just need to use the name of the class when calling the method (don’t need a new object first). It’s not as easy to tell method calls apart in instance methods because to call another instance method, you only need the method name (not an object declaration). This looks somewhat similar to the call for a static method, which would be the same but just with the class name and a period at the start of the call to the method. Also, in an instance method, other instance methods are called by referencing the object that called the original instance method. In order to make the code more clear, you can add the this keyword.

**R8.19. What is the this reference? Why would you use it?**

The this keyword is used as a reference to the object of class which it is in. When this is called, it refers to the variable in the same class which it is called in. The this reference can be especially useful in methods with parameters that have the same name as instance variables in the same class. In cases like these, you can use the this reference with the name of the variable to refer to the instance variable and the name of the variable alone to refer to the parameter variable. The this reference is often used to make it clear what is an instance variable and what isn’t – overall, the this reference can help improve readability of code.