

PART 3 INTRODUCTION TO OBJECT ORIENTATED PROGRAMMING

CHAPTER 9 OBJECT ORIENTATED PROGRAMMING CONTINUED

IN THIS CHAPTER, YOU WILL LEARN HOW TO:

- Explain and develop custom class objects.
- Recognize and define concepts of encapsulation as it pertains to information hiding, ode reuse and accessor and mutator methods.
- Recognize and define concepts of polymorphism by overriding and overloading class methods.
- Recognize and define concepts of inheritance along with concrete and abstract classes.
- Apply driver, custom and built in classes into programs
- Describe the programming process as it relates to object oriented programming.

OBJECT ORIENTATED PROGRAMMING CONTINUED AND EXPANDED

In this chapter, we will expand upon what we have learned about object oriented programming. In chapter eight, we covered general OOP vocabulary along with fundamental concepts. Most of our previous discussion of object oriented programming included an existing class that was incorporated in our program. Although you'll probably use mostly preexisting classes in your program development, there will also be many occasions where you'll need to design your own class files. We will call the classes we create, custom classes since they are being designed based on a requirement we have in our program and will contain data and methods of our design.

Learning about creating custom classes can be a lot like learning a foreign language. Like with a foreign language, many times it's easier to read the language then it is to speak it. Using classes works much the same way. It is much easier to read and use

a class that's been designed for you that it is to create one from scratch. This is not without precedence and we've seen this in earlier chapters.

Programming Tip -Programs and general are always easier to understand if the source code has been created for you versus having to create the source code from scratch.

Along with the task of creating our own class objects, we will also go deeper into encapsulation, polymorphism and inheritance. Unlike structured programming where we could focus more on logic and pseudo code, object oriented programming requires you understand concepts fundamental to code reuse and abstraction. In this chapter, we will continue with more examples and discussion to build on this.

INHERITANCE EXPANDED

From the last chapter, you will remember that inheritance can be defined as the ability of a child class to inherit data and methods from its parent. Because of this relationship, the children always have more class members than their parents and therefore in effect are more powerful than their parents. Inheritance can be a very powerful method of implementing code reuse by allowing data and methods to be shared generically across several child classes.

As a new programmer, it may be hard to see the importance of inheritance in code reuse at first glance. One of the most powerful statements about the power of code reuse provided by inheritance can be found in documentation of classes. Most programming languages (I will use Java in my example) come with extensive libraries of classes ready for program use. One of the pieces of information supplied with the documentation is the name of each class inherited by that class. As you look through the classes, you will be hard pressed to find an example where one of these classes does not inherit two or more classes. The professional OOP programmer needs to correctly apply inheritance to his classes and fully understand how to apply techniques of code reuse with inheritance.

In this chapter, most of our examples will surround the creation of a Student class used to represent the data and the methods associated with a college student. We will create several programs which will store and retrieve Student class data along with calling methods from the student class. The Student class was chosen because it's something that everyone can relate to and lends itself well to showing examples of encapsulation, inheritance and polymorphism.

DESIGNING CLASSES TO FACILITATE INHERITANCE

One of the first activities necessary in developing our Student class is to determine if any part of that class could be inherited from a parent class. If we think about a typical student for just a second, there are some attributes of Student which are shared by all students. A student has a name, a place of residence, a date of birth, a GPA, a major (among other things). If we were writing programs for a college or university, it might also be helpful to look at some other classes that will also be used with the Student class. A Faculty member and college President might also be made into classes and are also likely to be used with programs that use the Student class. The Faculty class shares many of same attributes and for that matter so does a college president. Do not both the Faculty member and President also have a name, a place of residence and a date of birth? If we were to create a class for each of these three "things", we would find that despite each being different that they all could belong to another class called Person if Person contained a name, a place of residence and a date of birth. Therefore, we should be able to create a parent class called Person that contains the name, residence and date of birth and then inherit that class into child classes that would reflect a Student, a Faculty member and a college President.

This would take care of any data that we shared across these classes but what about methods. Are there any actions that could be described as methods, in the person class and inherited by the student, faculty and president class? In this case, methods to be shared by these child classes may not be as easy to identify. There is one method that is common to most classes called the toString() method. The toString() method is special in that returns a string value describing the class. It is text that the class developer has determined to describe the class.. In this case, the toString() method will return the person's name, their residence and their date of birth as a string.

Programming Tip: Most class objects will have a toString() method defined. This is standard across object oriented programming languages. toString() is to be used as a helper method to return data as a string that can be displayed by the programmer. You can use this method as the programmer to help display information on the class or if the situation arises include it as part of your program.

THE PARENT CLASS

The parent class (also known as the superclass and base class) contains data and methods which will be inherited by the child class. It's important that this class remains as generic as possible to be inherited by a large number of child classes. If the members of this class become too specific, the classes' ability to contain reusable code will be diminished. For example, we might think that since we have more students than faculty members that more student data should be placed in the Person class. We might be inclined to add GPA to the Person class. If this was done the Faculty and President class could no longer inherit person since GPA data is not needed in their classes. Examine the pseudo code below and see the data and methods associated with a class named Person.

```
class person
  char name
  char street
  char city
  char state
  char zip
  char DateOfBirth

  toString()
    char outString
    outString = name + " lives at" street + city + state
+
    " and born on" + DateOfBirth
  return
end class
```

- The start of the parent class instance variables
- The start of parent class methods.

This person class contains a number of data elements that can easily be shared and a method that can generically provide a method to display class data. Are there more data attributes and methods that might also be added to person and not limit its ability to act as a parent class? Absolutely, but we are going to keep our class files smaller for learning purposes.

Parent and Child classes - With inheritance, one class acts as the parent and gives its class members to its children. The parent class and child class are referred to by different names in different languages. The parent class is called superclass and the child is called the subclass in Java. In Visual Basic.Net, the parent class is called the base class and the child class is called the derived class.

Programming Tip - A parent class can have several child classes. In our example, person has child classes of student, faculty and president. Child classes could also have their own child classes. Student might have child classes of inStateStudent and outOfStateStudent.

THE CHILD CLASS

The first thing you'll notice in a child class (also known as subclass and derived class) is that included in the class statement will be a reference to the inherited parent class. Different languages have different firms to designate the inheritance. We will use Java keyword extends and are pseudo code examples.

```
class Student extends Person
    num gpa
    char major
    num totalCredits
    char studentID
    DeansList()
        status = "false"
        if gpa > 3.25 then
            status = true
        else
```

```
        status = false
    return status
```

- Extends identifies the parent being inherited.

With this child class, we inherited all of what was included in our parent plus I have added four new data elements and one new method. Since the child has these additional members, it is said that child will always be more functional than the parent.

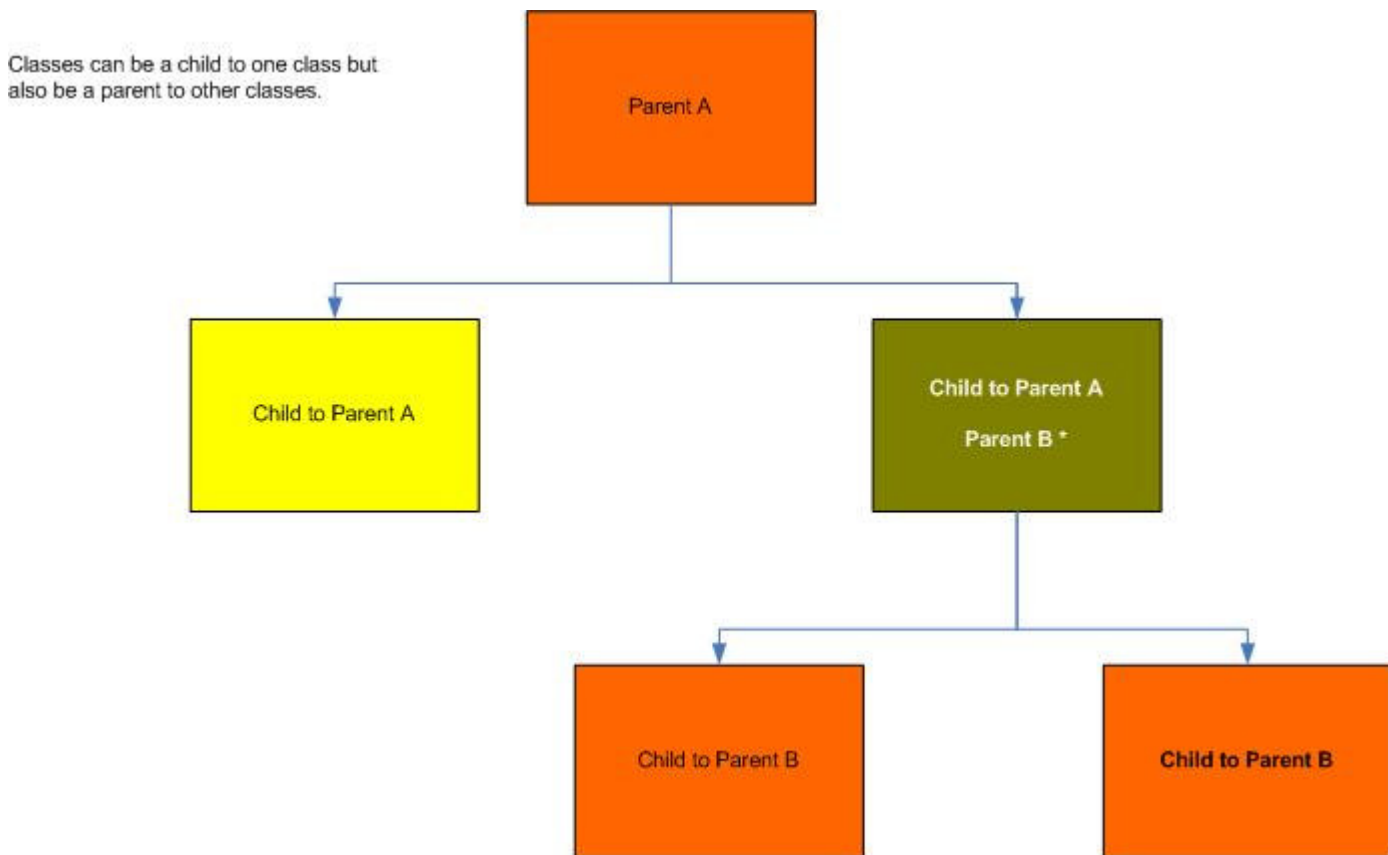


Figure 1: Parent - Child relationship

If we were to take this example one step further in use our parent person class with Faculty, you will see how the Person class can also be used to build a class object called Faculty. Below I have created a Faculty class that extends person.

```

class Faculty extends Person
    num dept
    char officeLocation
    char credentials
    char deptChair
    FacultyInfo()
        char outString
        status = super.toString() + officeLocation + dept
            credentials
    return status
end class

```

- Many times with creating a method you might want to use a method from the parent inside a method defined on the child. I am using the super keyword (this is used in Java) to indicate I would like to take the return value of the parents toString() methods and add to it data fields contained in the child record.

Status Check

Can a child class have its own children? If so, why would this be useful?

What is the difference between a base class and a superclass?

MORE ON POLYMORPHISM

Polymorphism can be a very broad topic in object orientated programming. In this wiki, we will focus our discussion of polymorphism on how it affects methods. The advantages to the programmer who uses polymorphism might be subtle. It is important to take the time to work with overloading and overriding methods to fully appreciate how polymorphism makes programs more powerful and flexible. Since polymorphism allows different forms of the same method to be included inside class files, it is now possible to invoke methods at runtime based on parameters or child classes' instantiated. This is also called dynamic binding and in this section of the chapter we will explain on how this is implemented.

OVERLOADING METHODS

The overloading of methods is a technique that is not unique to object oriented programming. Many programming languages allow the programmer to create different modules with the same name providing that the modules have a different input parameter list. As we start to implement both polymorphism and

encapsulation, the programmer has to pay special attention to what is called the method header. The method header typically contains the method scope modifier, the method name, the data type of the returned value in a parameter list indicating if one or more values will be passed into the method. In the case of overloading, and overloaded method is one which has the same name but a different input parameter list. Maybe one of the best examples of this behavior can be demonstrated in the calculation of the area of a rectangle. The area of our rectangle is calculated by multiplying the length times the width. A simple module to create unless the data being passed to the method needs to accept a different number of parameters with the parameters having different data types.

Unlike pseudo-code and PYTHON which has only one data type to represent numbers, most programming languages actually have several data types that support everything from simple integers to complex decimal numbers. As a result, it is entirely possible that our rectangle formula may require two integers, two decimal numbers, one decimal and one integer, or one integer and one decimal as input parameters to a module that calculates the area of a rectangle.

Area = 10 * 10 Area = 10 * 10.6

Area = 10.6 * 10 Area = 10.6 * 10.6

Table 1: Area calculations using different types of numbers (integer and decimal)

With overloading, it's possible for us to create one module called area and duplicate it four times to accommodate all of the different data types that may be possible to the within the modules parameter list. Look at the following pseudo code. This pseudo code example has extended the number data type from just num to integer and decimal.

```
integer length_int
integer width_dec
decimal length_dec
decimal width_dec
area(length_int, width_int)
    integer myArea
    myArea = length_int * width_int
return myArea
```

```

area(length_int, width_dec)
    decimal myArea
    myArea = length_int * width_dec
return myArea
area(length_dec, width_int)
    decimal myArea
    myArea = length_dec * width_int
return myArea
area(length_dec, width_dec)
    decimal myArea
    myArea = length_dec * width_dec
return myArea

```

- area method with two integers are input parameters.
- area method with one integer and one decimal as input parameters.
- area method with one decimal and one integer as input parameters. Note how the order is now different. Before we had integer, decimal and now we have decimal, integer.
- area method with two decimals are input parameters.

Depending on the values that are sent to this program by the user, one of the overloaded area methods will be called. Which one is called depends on the users input. The method is decided at run-time based on the inputs. If it wasn't for overloading, how might your program have built differently to accommodate the same flexibility? More than likely a series of decisions structures (if statements) would have to be constructed to determine what the data type was of the values input and then one of four differently named methods called based on the testing of that input.

OVERRIDING METHODS

Yet another implementation of polymorphism is overriding. Like overloading, overriding gives you flexibility as to which method executes at runtime. For overriding to take place, a class must first inherit a parent class. The next step requires that the child class override one or more of the parent's methods. Overriding is done by creating a duplicate method header in the child class but with a different implementation. Implementation would be those statements inside the body of the module. For example, I have decided to create two child classes of student named A1CollegeStudent and B1UniversityStudent. Both of these classes will inherit all the members of Student and include some new members that are unique to each of these new classes. Since each school has a deans list but each school's deans list is determined by different GPA, the A1CollegeStudent will to override the deansList method of Student to reflect the fact that this school has a requirement for the deans list GPA to be 3.5 and above. Since B1UniversityStudent has the same deans list logic as Student, it does not have to be overridden.

See below for pseudo code that demonstrates overriding.

Parent Class with DeansList method

```
class Student extends Person
    num gpa
    char major
    num totalCredits
    char studentID
    DeansList()
        status = "false"
        if gpa > 3.25 then
            status = true
        else
            status = false
    return status
```

Child Class with DeansList method overridden

```
class A1CollegeStudent extends Student

    char campusLocation
    num financialAid

    DeansList()
```

```
status = "false"
if gpa > 3.5 then
    status = true
else
    status = false
return status
```

- Here we have new class members specific to A1CollegeStudent

- Here we have overridden methods that will use different logic that what was inherited from Student.

Since UniversityStudent does not have different logic concerning the designation of deans list, it does not need to implement an overridden version of deansList.

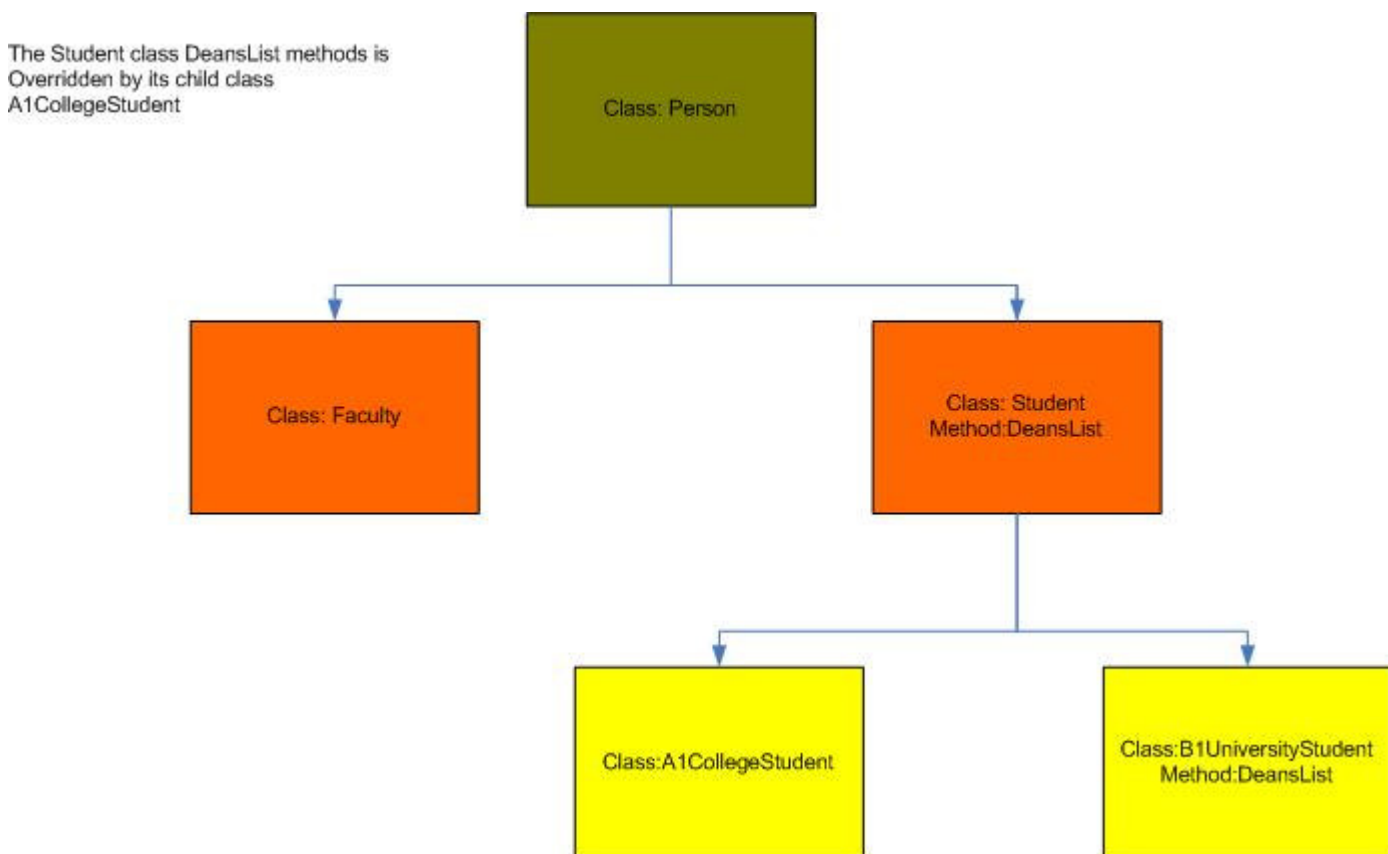


Figure 2: The object hierarchy showing the DeansList method overridden by its child class.

Status Check

Why is overriding related to inheritance?

Give the conditions in the parameter list that allows for more than one method with the same name?

Background Information: Object Hierarchy - Object hierarchy describes the family tree of the class object. For example, if we look at the A1CollegeStgudent class we will see that its object hierarchy includes a parent called Student. Student also has a parent called Person. Because of this family relationship, A1CollegeStudent gets everything up the family tree including the members of Person. Most programming languages provide documentation that outlines what classes have been inherited by a class. This can provide valuable information in determining what class members are available to your program.

Object Hierarchy - The object hierarchy acts like a family tree for a class by showing all of the classes it inherits members from. Class object documentation should include an object hierarchy for programmers using your classes.

ENCAPSULATION EXPANDED

You will remember from chapter eight we defined Encapsulation as "information hiding." To fully implement Encapsulation we would have to tie the scope identifiers private and public to our methods and instance variables. We would use public methods to set and get data stored in the instance variables. We want to protect the instance variables from outside calls and applications. This protection would include not allowing invalid data from being stored in the object (i.e. cost values must have a value greater than zero). We also want to hide details of the class since details are not important to the class user. The class user only needs to know how to communicate with the class via its methods.

Encapsulation is implemented via the public mutator and accessor methods which protect are class data. By convention, set methods are mutator methods and allow us to update instance variable values. Get methods are accessor methods in allows

us to retrieve information stored inside the instance variable. All communication and updating with instance variables is accomplished through get and set methods..

Private scope - Private variables and methods are visible to class members only. Private members can not be accessed by other classes (including driver classes).

Public scope - Public variables and methods are visible to class members and all other classes with access to the class.

Programming Tip: Encapsulation is important in supporting code reuse but in fact is optional in most programming languages. In Java for example, one could create a class for reuse across to many Java applications and not build within the code a structure that supporting cancellation. Although possible, this is not advisable because without encapsulation more information needs to be provided to the programmer which makes reuse less feasible.

ACCESSOR (SET) AND MUTATOR (SET) METHODS EXPLAINED

Most languages use accessor and mutator methods along with private and public scope of identifiers to ensure data hiding in class objects. This is implemented by creating methods of public scope the provide access to data (also known as instance variables) that have private access. The public methods are divided into two categories. One is the mutator or "set" method and the other is an accessor or "get" method. The "set" method is used to change the values in the private instance variables and the get method is used to retrieve the data stored in the instance variables. For both of these public methods, you should by convention prefix the method with "get" for accessor methods and "set" for the mutator methods. The prefix is placed in front of the name of the data instance variable (i.e. the private customerName instance variable would have a public accessor method called setCustomerName and a public mutator method called getCustomerName).

Accessor methods - Accessor methods are modules with public scope that allow for the retrieval of private instance variables. Accessor methods are important in supporting encapsulation

Mutator methods - are modules with public scope that allow for the update of private instance variables. Mutator methods are important in supporting encapsulation

Encapsulation – “information hiding”

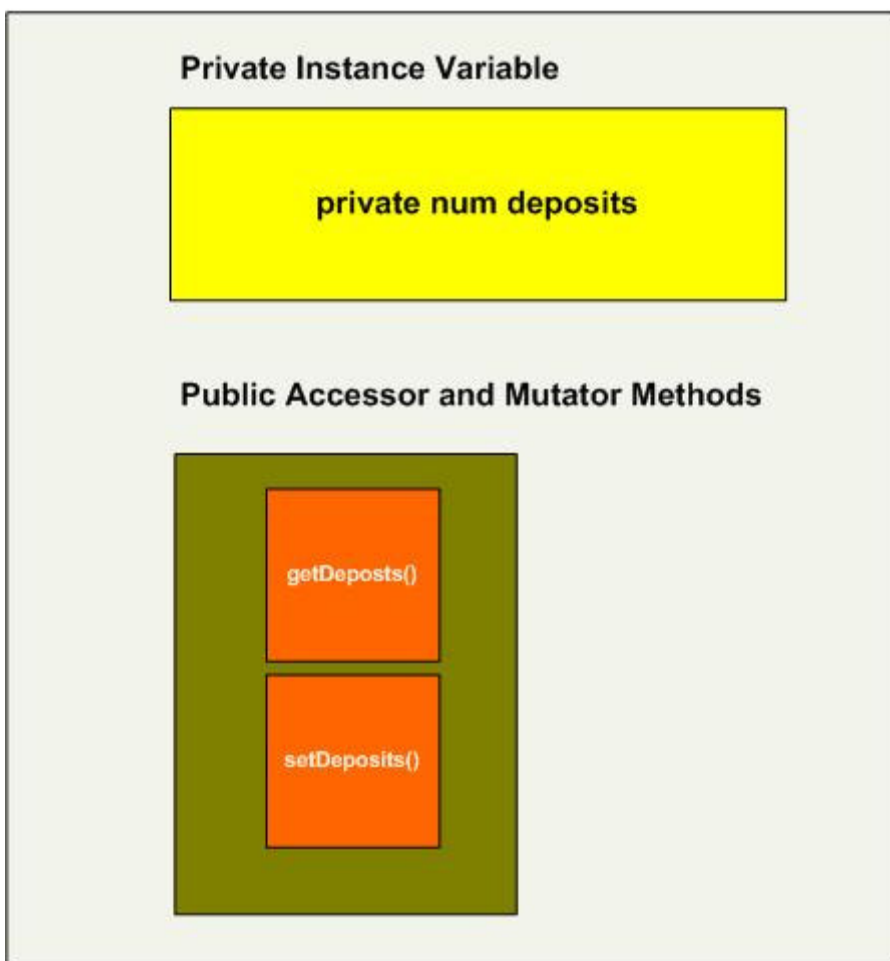


Figure 3: Encapsulation with get and set methods shown graphically

PSEUDO CODE EXAMPLE OF ENCAPSULATION

Below is an example of private instance variable (deposits) with public accessor (getDeposits) and mutator (setDeposits) methods

```
private num deposits = 0
public setDeposits(inVal)
    if inVal > 0 then
```

```
        deposits = inVal
    else
        print "Invalid Input - Must be greater than 0"
return
```

```
public getDeposits()
    return self.deposits
```

- The mutator set method protects the deposits instance variable from invalid data by displaying an error message when invalid data is sent to the class member.

- The accessor get method retrieves and returns back to the class the data stored in the deposit instance variable.

Related Subject: OOP and program patterns - Many times this wiki we've talked about programming patterns. These patterns are blocks of program logic which because of their generic utility can be used over and over and in a variety of applications. The logic necessary to read through the records of sequential files example of a programming pattern used in the previous chapter. The same patterns can also exist in class objects. Just like the structured programming modules, object oriented methods can also contain statements which would qualify as patterns that could be reused over and over again. In fact, due to the enhanced reusability of OOP, class objects are a perfect way to implement code patterns. Many OOP languages (i.e. if the Java and Microsoft .NET) provide API (application program interface) documentation that documents reusable classes that contain many of the programming logic the patterns identified in this wiki.

Status Check

Are accessor and mutator prefixes of get and set required? If not, what are they?

Why is identifier and method scope important in support encapsulation?

USING EXISTING CLASS OBJECTS

Programmers use class objects from a variety of sources. Typically, most programming languages contain extensive class libraries to cover most of the common routines (patterns) necessary to build application programs. Building graphical user interfaces that incorporate controls like command buttons and text

boxes to classes that facilitate opening and closing text files. All of this accomplished with class objects. For most OOP languages, classes are divided into namespaces (also referred to as packages). A namespace or package acts like a folder that holds classes that provide a similar function. For example, many languages support an IO (stands for input output) namespace that contains classes to facilitate file processing. Just a programmer can create their own classes; they can also create their own namespaces. It is recommended that custom namespaces include a URL associates with the author as part of the namespace name to avoid name duplication.

Namespace / Package - Namespace and package files act as containers to hold classes that perform similar functions. They make classes easier to locate and use along with help to uniquely divide classes since the package name becomes part of the reference to the class.

You can identify a namespace as a prefix parameter in the class name or you can use the import keyword to associate the namespace with the class and make programming statements easier to write. Below I have two blocks of code that access a custom class. One state is called without the imports keyword and the other does not.

garymarrer.myclasses.minMaxClass.min(numOne, numTwo) or with an import statement

```
imports garymarrer.myclasses.*
minMaxClass.min(numOne, numTwo)
```

- This example comes from Java although other languages use similar syntax. Notice how the reference to the class name is shorter in this code block because the package has been identified in the imports statement

CUSTOM CLASSES

For those of you will be taking Visual ~Basic.Net or Java for your first programming class, you'll find that there are two types of custom classes you'll need to work with. The first is the driver class which is used to control your program. The second customer classes are the classes that will contain the data and methods to describe "things" you are interested in sharing across programs. With both driver classes and the custom classes you are creating, you will include within them calls to other classes. Some classes which will come from class libraries (with numerous namespaces) included with the language and others you may have created. The programs in OOP become a mix of various class objects and control logic.

DRIVER CLASSES - YOUR FIRST CUSTOM CLASS

The first custom class is a driver class and typically the first class you will create as an object oriented programmer. The driver class contains the instructions that determine the order in sequence of the program logic. The driver class looks and acts very much like the programs we developed in structured programming. The driver class still contains instance class variables along with methods but acts as a self contained unit where all the program logic could exist within that one class. The driver class holds a special status and that it can be executed as a program. The drive class is the first class the executing environment looks for to start the program. I would contrast this with other custom classes which contain only data and information about anything used within the program. These classes need a driver classes to instantiate them into call the members during program execution. For example, the Student class we worked on earlier in the section on inheritance can not be executed unless it is instantiated and called in a driver class (note: there are some exceptions to this but any valid program must be started by the driver class).

Key Concept: In most object orientated programming languages, the main logic of the program is also kept inside a class file even though this class may not be instantiated. In Java for example, the driver class has a class statement that corresponds to the program name. When the program name is executed the driver class is found and the logic of the driver class is started. This makes the driver class very similar to our program files created using structured programming techniques.

Driver Class - The driver class contains the mainline logic which drives the instantiation of other class objects and uses class objects with programming structures (i.e. control structures) to define the logic of the program and the sequence of the called instructions. The execution environment looks for and starts with the driver class.

Logic TIP: The driver class is probably the closest thing to the logic found in a structured program. It contains the mainline logic necessary to drive the steps of the programs.

CLASS SCOPE

Just as variables have program scope so do class objects. The scope of classes is much the same as is with variables. When the program instantiates a class object, that class object points to a location in memory where the class object has been loaded. If the object variable has been instantiated and inside a method then that class is local to that method and not available to other methods in the driver class. If the class object is declared in the body of the driver class, this makes the class and its members available to any statement and any method of the driver class. See below for example of a class that's declared as global to the driver class (globalStudent) and a class that is declared local (localStudent) to a method.

```
public class myTest

private Student globalStudent

public myMethod

private Student localStudent
    print globalStudent.toString()
    return
end class
```

- globalStudent represents a custom class that because it has been defined in the body of the class is global to all of the class members. As a result, the globalStudent class object is callable from within the myMethod method.

- myMethod contains its own local class and also calls a global class

- localStudent is only visible to the statements in myMethod.

- globalStudent is visible because the object variable was defined in the body of the driver class

COMPILED CLASSES

For instruction purposes, we have always shown the class files source code called by the driver class. This is a luxury that you typically will not have as a programmer. For most of the classes that either are called from the programming language libraries or custom classes written by other programmers, you'll we'll have access to the compiled code only. When a program or class is compiled the source code statements, which are typically in English and formatted with the programs keywords, will be converted to either machine code or byte code which consists of binary characters. Binary data cannot be opened with a text editor to inspect programming statements. What this means is that you will have to understand what each class contains in how to call its members. This information is available in a couple of places. First of all, many languages provide a software application via the IDE called the class browser. The class browser acts like file manager works with files except the class browser allows you to look at class members. With a class browser, you can see what a class member will return in what inputs are required as parameters.

Still another source of information about the documentation of classes and its members is through the API (Application Program Interface) documentation. This information will be displayed in a more traditional format where the class and its members will be cross reference to a table of contents which will allow you to navigate to the information you're interested in finding out about. From that information, you will be able to find out the name of the class and how to use each of its members.

Class Browser - A class browser is a software tool included with the programming languages IDE to display the classes used by the program and the members of those classes. The class browser will let you browse through your class objects like a file browser lets you browse through your hard disk...

API (Application Programming Interface) - The API is the documentation included with your classes to identify the classes in the package and the members in each

class. The API is similar to the class browser but is in the form of documentation whereas the class browser is a software application.

You may be asking what the benefit of a compiled class is if it requires the programmer to access these other information sources to know how to use the class object; for starters many classes are very complex and written by experienced programmers. The even if the source code was available, the average programmer may not be able to glean any effective information from looking at the source code. After all, is this not the reason for information hiding and code reuse? As we have discussed in previous topics, the programmer need only know what messages to send the classes input and understand what messages will be returned as output to effectively use a class and its members.

Still another reason for compiling class objects has to do with security and protecting intellectual property. It may be entirely possible that certain classes execute code which if available to less scrupulous programmers, might allow for the hacking or misuse of the class. a compiled binaries class object minimizes this exposure. A second and probably more plausible benefit for compiling class objects has to do with the protection of intellectual property. As a programmer, your source code statements are just as valuable to you as the lines of text to an offer writing a novel. The programming instructions represent the fruits of your labor and the design that you alone have created. For this reason, this design in the implementation of it is yours to own and should not be available to others to use or exploit without your permission. Although not foolproof, compiling a class object will help minimize any theft of your intellectual endeavors regarding the custom class you created.

Compiling class objects can also help with program maintenance and reuse. If the source code is delivered with the class any programmer could modify the source code and create a new version of the class. One of the requirements of code reuse is that reusable code is the same everywhere it is used. If original class was changed, it would be unlikely that the change made by the original author would work correctly on any classes that were modified without their knowledge. Think about this scenario. What if I was to send a class object out to my programmers that contained the calculation of product cost? One of the programmers using the class changed the class object directly instead of inheriting it. I then send an update to

reflect a new product cost. There is a strong possibility that the changed class code would break unless they again changed my original class code. This is unproductive and sloppy most likely leading to program quality issues. Compiled classes prevent unsanctioned updates of the class by hiding the source code and requiring programmers using the class to interface with it or inherit it into a new class.

METHOD HEADERS

As the previous two topics elaborated on, most of the time as a programmer choosing to use classes to implement a solution, you will not see the source code. You will work with the class via messages. Because of this, the class browser in any documentation available for the class will only give to you information necessary to send inputs and received an output from that member. This is especially true with class methods where each method has a name, a parameter list and an indication if any data is returned by the method.

For example if we look at the set and get methods used earlier in regards to demonstrating encapsulation and on the deposit instance variable, we will see that the setDeposits method has a header which requires one character value as input without returning any valued back to the calling statement. The method header for getDeposits method accepts no parameter information as input but does return a single value. What happens inside these methods is of little consequence to the programmer. The programmer trusts that the person who created this class member has code the logic of the member correctly.

```
setDeposits(num inVal)
    deposits = inVal
return
```

```
getDeposits():
return self.deposits
```

Class Method Headers

Java deansList Method

```
public boolean deansList(double gpa)
{
    if (gpa > 3.5)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Header Information:

Name: deansList

Input parameters: one, data type double

Returned variables: one of data type boolean

Visual Basic.Net DeansList Method

```
public DeansList( gpa as decimal) as boolean
{
    if (gpa > 3.5)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Header Information:

Name: deansList

Input parameters: one, data type double

Returned variables: one of data type boolean

Figure 4: Java and Visual Basic.Net Method Header

ABSTRACT CLASSES AND ABSTRACT METHODS

To describe the concept of abstract classes we need to return to an example used earlier when we discussed inheritance. We had a scenario particular to the parent class of Student. It turned out that the Student class was inherited by two subclasses, specifically, A1CollegeStudent and B1UniversityStudent. Even though many of the members could be inherited and utilized easily by the subclasses, it was at least one method called DeansList which had to be overwritten by the A1CollegeStudent class because its rules regarding the DeansList were different than its parent. In the case of, B1UniversityStudent, the DeansList method did not need to be overwritten because it shared the same criteria as its parent student.

Abstract classes give us more flexibility in regards to situations like what was seen with the DeansList method. Perhaps a more efficient and more reusable solution would have been to create an abstract method called DeansList. An abstract method is a method that contains only method header information. With abstract members, it is expected that the child class will implement that method. Implement in object oriented programming means create the body of the method. In other words, the parent's abstract method contains only the header line of the method in the child's class contains that same header line plus all the statements that would go with that method. If this was done on the DeansList method in the previous example, both A1CollegeStudent and B1UniversityStudent have their implementation of DeansList. The pseudo code below illustrates how this would look.

Abstract Class - Is a class that can not be instantiated because one or more of its members are abstract and contain no class body. Abstract classes can only be inherited.

Programming Tip- If you have an abstract method but have not defined the class as abstract then most compilers will return a compiler error message.

Parent Class

The parent class contains an abstract method which will require that the class be designated as abstract.

```
class abstract Student extends Person
    num gpa
    char major
    num totalCredits
    char studentID

    abstract DeansList()
```

- Since the class contains an abstract method, the entire class needs to be identified as abstract.

- The abstract class DeansList only contains header information.

Child Classes

Both child classes have been represented in the pseudo code below. Note how both classes inherit Student and both have their own implementations of DeansList with the same header as their parent (Student) but with different logic in the body of the method.

Child Class 1

```
class A2CollegeStudent extends Student
    char campusLocation
    num financialAid

    DeansList()
        status = "false"
        if gpa > 3.5 then
            status = true
        else
            status = false
    return status
```

- Here we have the abstract DeansList method implemented in its subclass.

Child Class 2

```
class B1UniversityStudent extends Student
    char campusLocation
    num financialAid

    DeansList()
        status = "false"
        if gpa > 3.5 then
            status = true
        else
            status = false
    return status
```

- Here we have DeanList method implemented with logic that requires a student have a GPA greater than 3.5.

CONCRETE CLASSES

Until we introduced abstract classes we were working exclusively with concrete classes. Concrete classes contain no abstract methods and can be instantiated as and used as class objects in our programs. As mentioned earlier, abstract classes can only be inherited and not be instantiated as a class object. Common sense dictates the reason. These classes are incomplete due to the fact that one or more of their methods are not fully implemented.

Concrete classes: Concrete classes are the default class and have no abstract class methods. They can be inherited or instantiated.

TESTING OOP PROJECTS

Testing programs developed using object oriented programming languages is fundamentally no different than what was discussed earlier we were developing our programs using structured programming techniques. With any good test is a plan, as the programmer, you have to ensure that all the logic within your program is tested appropriately. The test plan should systematically identify all modules and control

structures and test all inputs that would cause the program to fail or not return a useful error message.

Status Check

Why are abstract class only inherited?

Why is it that the method header is all a programmer needs to use a class method?
is this an example of abstraction or polymorphism?

OOP AND THE PROGRAMMING PROCESS

In chapter one, we discussed how programmers use a process to develop programs. We used this process when we created programs using structured programming techniques and now we will review this process again in light of what we have learned with object oriented programming. You will see that the process stays the same and can be used just as effectively with OOP as with structured programming. From a prior chapter the definition of a program development process:

Program Development Process (PDP) - you can think of the program development process as standardized set of steps used to provide a logical, common sense approach (or process) to solving a difficult problem with a computer program. PDP is for the programmer a lot like a checklist that ensures that the steps of building a program are followed so that no important step is omitted or taken out of order. This process has long been used by programmers as a professional standard because of its simplicity and consistency.

This program process is still relevant to the programmer using object oriented techniques even if some of the modeling tools are different.

- Start with a **problem statement**: No change here. This step stays the same and is critical in defining the scope of the problem and what exactly your program will need to do.

- **Fact finding** or requirements analysis (done by interviewing program end user): Fact finding is still critical to capture the background data necessary for identification of classes, class members and logic for the driver class.
- **Conceptual design:** Conceptual design usually takes the form of pseudo code or a flowchart that lays out the ideal logic structures to provide a solution. Our conceptual design has focused on pseudo code in the past two chapters. Suspiciously absent were flowcharts which will be replaced with UML (Unified Modeling Language) in the next chapter. UML is the standard for graphically modeling logic that will be implemented using object oriented programming design.
- **Detail design:** We are ready to code our solution using a programming language. We take the conceptual design as a starting point and using the syntax options of the programming language, code the logic structures into programming language structures. This process step also does not change significantly except we are now designing driver and custom classes in addition to just program logic statements.
- **Testing:** We will perform unit tests as we go. We always need to test and the testing of class objects is just as important as the testing of structured programming applications.
- **Implementation:** During implementation, we move into production the test average calculator program on to the PC of the person using it. The program is complete and ready to use.
- **Maintenance:** No program is written once and not revisited without some changes. With object orientated programming and its extensive use of reusable code, maintenance is made more efficient since a class code is shared and a change to that class can be made once and propagated across to all the programs that use the class object.

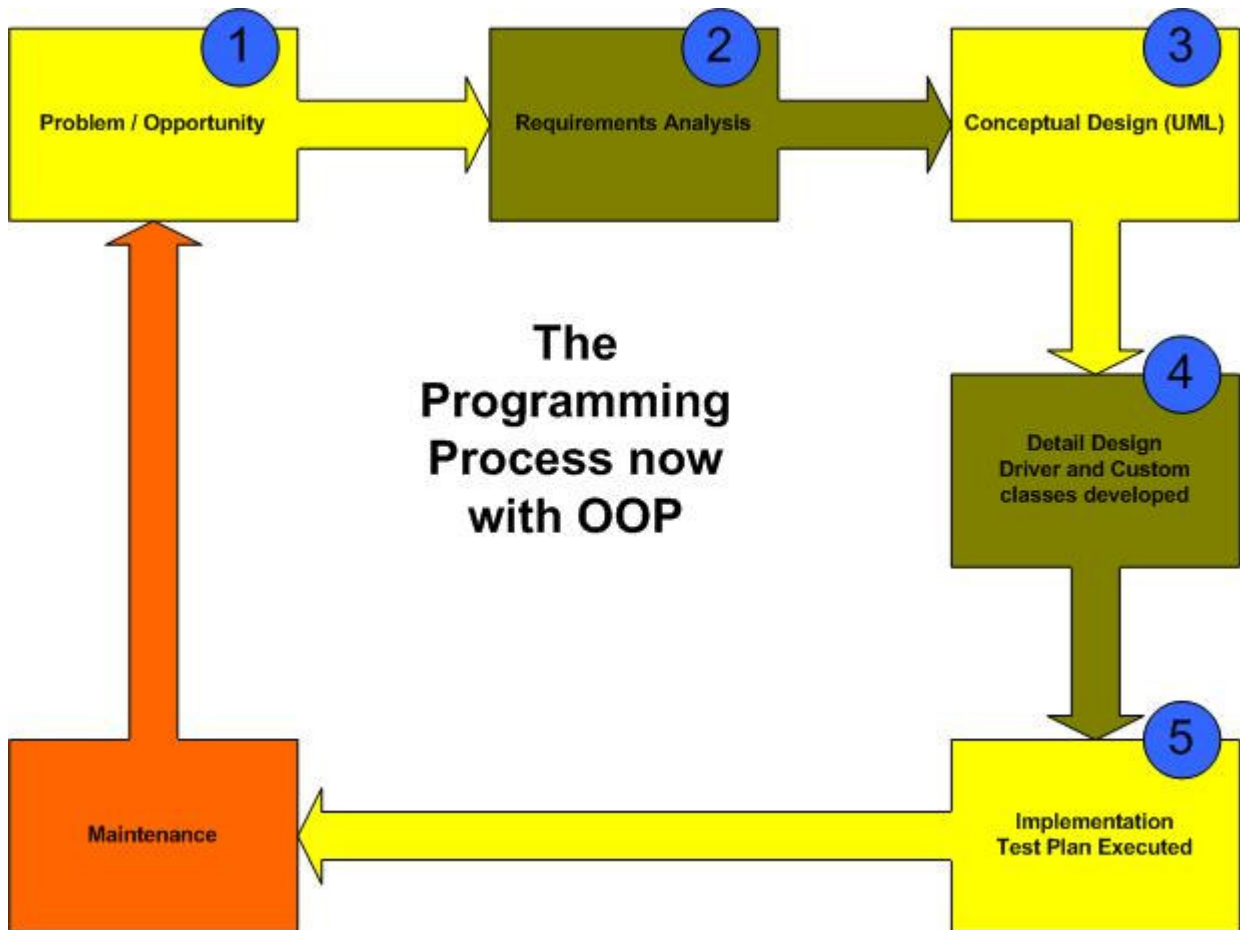


Figure 5: The programming process using object orientated design techniques

CACTUS CASE STUDY

The light bulb is starting to get a little brighter with his chapter's coverage of object oriented programming. You are beginning to see how OOP can make a significant difference in programmer productivity and how classes can save a lot of time and maintenance down the road. Based on Cactus Books and Tapes web site requirements, the web payment service can be implemented in several programs. But more importantly, now that you understand more about inheritance, the WebService payment method can be inherited and expanded upon to fill in added function needed by the web site but not included in the web service.

You have told the Chavez sisters that reusing the code from the credit card company and adding an overriding the class with additional members can accommodate many of the e-commerce functions needed to keep the web site secure and easy to use for the customer.

Look at the following requirements and implement then into a new class called CBTPayment. You will need to create a driver class to test your new class members. To do this your program must input parameters to each of the new methods for testing. We only work on testing the class in this exercise. We need to show the Chavez sisters how this is going to work.

CASE STUDY TASKS:

- Inherit Web Payment into a new class called CBTPayment (CBT for Cactus Books and Tapes)
- Override Payment to capture store number and payment date
- Add Instance Variables for store number and payment date
- Add set and get methods for new instance variables
- Create Driver Class to test the new class

WEB SERVICE CLASS OBJECT WITH PAYMENT METHOD

This is the class called from the credit company.

```
class WebPayment
    payment(customerName, customerCCNumber, expDate, Amount )
        open CustomerFile
        while not End-Of-File
            read custID, custCC, custExpdate
            if customerName = custID and
            custCC = customerCCNumber and
            custExpDate = expDate then
                status = "approved"
            else
```

```

        status = "rejected"
    return status
end class

```

WEB SERVICE CLASS OBJECT WITH PAYMENT METHOD

Below is our new class with new instance variables, inheritance, encapsulation and polymorphism implemented.

class CBTPayment inherits WebPayment

```

char storeNumber
char transDate

payment(customerName, customerCCNumber, expDate, Amount)
    open CustomerFile
    while not End-Of-File
        read custID, custCC, custExpdate
        if customerName = custID and
           custCC = customerCCNumber and
           custExpDate = expDate then
            status = "approved

                setTransDate(transDate)
                setStoreNumber(5)
            else
                status = "rejected"
    return status

setStoreNumber(inStoreNumber)
    if inStoreNumber <> 5 then
        storeNumber = inStoreNumber
    else
        print "Invalid Internet Store Number"

return
getStoreNumber()
return storeNumber

setTransDate(inTransDate)
    if inTransDate <> "" then
        transDate = inTransDate
    else
        print "Date can not be blank"

return
getTransDate()

```

```
        return transDate
    end class
```

- The CBTPayment class inherits the WebPayment class with the inherits keyword

- By convention, instance variables are defined at the top of the class statements. You will see each instance variable has its own set of mutator (set) and accessor (get) methods.

- The payment method has been overridden by defining a method with the same name

- We have changed the payment method by collecting the transaction date and store number with this internet transaction. The store number will be hard coded to number 5 indicating this transaction was done via the Internet.

- The mutator setStoreNumber method will only allow a 5 to be stored as a store number.

- The mutator setTransDate methods will not allow a blank or empty transaction date.

Pseudo code Driver Class Solution

Note: For this program, I have left off the robust data validation you would normally find in your program. We are focusing using a class and calling its members in this example.

```
class MyClassTest
    declare Variables
        CBTPayment myPayment
    custName = input("Enter Customer Name ")
    custCreditCard = input("Enter Customer Card Number ")
    custExpDate = input("Enter Card Expiration Date ")

    custAmount = input("Enter Purchase Amount ")
    storeNumber = input ("Enter Store Number")
```

```
transDate = todaysDate()

print "Purchase Status"
print myPayment.payment(custName, custCardNumber, custExpDate ,
custAmount)
print "Thank you for your Business!?"
end
```

- Here we have two number instance variables called from our new class

- Most programming languages have a date function to extract from the computer todays date. todaysDate() is a placeholder indicating the programmer should use the built in function for today's date. Our example will use the built in today() function from PYTHON.

PUTTING IT TO USE

The intent of this section is to introduce the new programmer to standards and techniques frequently used by professional programmers. Topics in this section relate to concepts introduced in the chapter but with a more vocational or occupational focus for students considering a career in programming.

Best Practices: OOP Conventions and Standards - OOP is no different than structured programming in that there are standards which have been developed which need to be incorporated into the design and coding of OOP based applications and Classes. Creating applications which use industry standards will provide a convincing case to your professionalism in the development of software. It will also make maintaining your program much easier. You should learn in implement these standards in all of your OOP programs. The sooner you train yourself to use these standards, the better off you'll be as you create more complex programs.

A LIST OF STANDARDS COMMONLY USED IN OOP:

- To support encapsulation your mutator methods should start with the prefix set (lower case) and your accessor methods should begin with the prefix get (also lower case). The access methods should have public scope and the instance variables should have private scope We will cover get and set methods formally in the next chapter.
- Instance variables and methods should always start with a lower case letter. If the method or instance variable contains two or more words in every word after the first letter should start with an upper case letter, this is called camel casing. Instance variable are defined at the top of the class file before methods. For example, the variable total cost would be represented as totalCost. See how the C in cost has been capitalized.
- Class Definitions should always start with an upper case letter. By convention, class files always have the first letter capitalized. Other letters are a combination of lower and upper case but typically follow camel casing conventions.

ADDITIONAL INFORMATION: CODE MAINTENANCE WITH OOP DESIGNED PROGRAMS

I have already in this chapter and the previous chapter alluded to the fact that class objects can make code maintenance simpler with fewer chances for introducing new errors. It is best to present this as a scenario. For example if we go back to the Person and Student classes covered in the section on inheritance, what if I decided that each person also has a telephone number? If telephone number is truly something which belongs to a person, I can add telephone number to the Person class and it will automatically give it to all of my children. This is in contrast to deciding that each of the children needs a telephone number. If they did not inherit from their parent each child class would have to have code added to it to accommodate telephone number data. Not having telephone number inherited could be more time consuming and complicated because of the number of classes that would have to be updated. With many telephone number updates to make, the possibility of missing one is very high. Missing one on a child class is zero.

I have an example for you to think about. Does everyone remember the infamous Y2K melt-down scare where all computers would be coming to a screeching halt when the clock struck twelve and the date moved over to 1/1/2000?. Each program had to be checked for any invalid year 2000 date processing code. If the date code was inherited, would the verification and repair been as labor intensive (and therefore very time consuming and expensive)? The answer is no. Use of class

objects and inheritance would have certainly made class updates much more effective.

Background Information: One Error for Every Two Errors Fixed - It is not unusual for a fixed logic error to introduce new errors. I once worked on a project where for every two errors fixed a new error was introduced. Not a record to brag about and one that required an increased emphasis on program quality (a lot of late nights and status reports to management). This system did not use an object oriented programming environment and with code reuse I am confident that we would have seen much better quality. A library of reusable code would have brought a homogeneity and consistency that would have helped better manage error resolution.

CHAPTER REVIEW

Chapter summary, highlights, key terms, short answer questions, quizzes and case studies to reinforce topics covered in this chapter.

CHAPTER SUMMARY

After finishing this chapter you should understand and in some cases, demonstrate the following topics:

- Explain and develop custom class objects.
 - Although as a programmer you will probably make more use of predefined class objects, there will be many occasions where you will want to create class files to share across multiple programs because of the code reusability.
 - Many languages use create custom driver classes, which closely resemble the programs written with structured programming techniques, to hold the program's logic.
- Recognize and define concepts of encapsulation as it pertains to information hiding, code reuse and accessor and mutator methods.
 - Encapsulation is implemented by creating public scope accessor and mutator methods to update and retrieve private scope instance variable data.

- By industry standard, mutator methods are identified with the “set” prefix and the accessor methods are identified with the “get” prefix.
- Recognize and define concepts of polymorphism by overriding and overloading class methods.
 - Polymorphism provides dynamic binding for concrete and child classes by allowing methods to be overloaded or overridden.
 - When a method is overridden, the child class creates methods with the same method header as its parent but with different logic in the method body.
 - Overloaded methods allow for multiple methods to have the same name but different parameter lists. The program decides at runtime based on the parameters which version of the method to execute.
- Recognize and define concepts of inheritance along with concrete and abstract classes.
 - Inheritance allows the programmer to implement code reuse along with simplifying maintenance by sharing class members across class files.
 - Inheritance consists of a parent class and child class. Parent classes are also known as superclasses and base classes and child classes are also known as subclasses and derived classes.
 - Abstract classes contain one or more abstract methods that have a method header defined but no method body. Abstract classes can only be inherited where their children must implement the abstract members they inherited.
 - Classes that are not abstract are concrete and can be instantiated. Classes are concrete by default.
- Describe the programming process as it relates to object oriented programming.
 - The programming process with OOP stays the same as defined with structured programming and can be used just as effectively with OOP as with structured programming.

○ The conceptual design phase in object orientated programming moves away from flowchart and towards UML (Unified Modeling Language) which was developed specifically for object oriented design.

CHAPTER KEY TERMS

Abstract Classes

Accessor Methods

API (Application Program Interface)

Child Class

Class Browser

Concrete Classes

Driver Class

Mutator Methods

Namespace / Packages

Object Hierarchy

Parent Class

Private Scope

Public Scope