Object-Oriented Programming in Visual Basic
The words "class" and "object" are used so much in object-oriented programming that it is easy to confuse the terms. Generally speaking, a class is an abstract representation of something, whereas an object is a usable example of the thing the class represents. The one exception to this rule is shared class members, which are usable in both instances of a class and object variables declared as the type of the class.
Fields, Properties, Methods, & Events

Classes consist of fields, properties, methods, and events. Fields and properties represent information that an object contains. Fields are like variables because they can be read or set directly. For example, if you have an object named "Car" you could store its color in a field named "Color."
Properties are retrieved and set like fields, but are implemented by property Get and property Set procedures, which provide more control on how values are set or returned. The layer of indirection between the value being stored and the procedures that use this value helps isolate your data and lets you validate values before they are assigned or retrieved.
Methods represent actions that an object can perform. For example, a "Car" object could have "StartEngine," "Drive," and "Stop" methods. You define methods by adding procedures, either Sub routines or functions, to your class.
Events are notifications an object receives from, or transmits to, other objects or applications. Events enable objects to perform actions whenever a specific occurrence occurs. An example of an event for the "Car" class would be a "Check_Engine" event. Because Microsoft Windows is an event-driven operating system, events can come from other objects, applications, or user input such as mouse clicks or key presses.
Encapsulation, Inheritance, and Polymorphism

Fields, properties, methods, and events are only one half of the object-oriented programming equation. True object-oriented programming requires objects to support three qualities: encapsulation, inheritance, and polymorphism.
Encapsulation means that a group of related properties, methods, and other members are treated as a single unit or object. Objects can control how properties are changed and methods are executed. For example, an object can validate values before enabling property changes. Encapsulation also makes it easier to change your implementation at a later date by letting you hide implementation details of your objects, a practice called data hiding.
Encapsulation - Cont.

Encapsulation is the ability to contain and control access to a group of associated items. Classes provide one of the most common ways to encapsulate items. In the example below, the BankAccount class encapsulates the methods, fields, and properties that describe a bank account.
Encapsulation - Cont.

Without encapsulation, you would declare separate procedures and variables to store and manage information for the bank account, and it would be difficult for you to work with more than one bank account at a time. With encapsulation you can use the data and procedures in the BankAccount class as a unit. You can work with multiple bank accounts at the same time without confusion because each account is represented by a unique instance of the class.
Encapsulation also allows you to control how the data and procedures are used. You can use access modifiers, such as Private or Protected, to prevent outside procedures from executing class methods or reading and modifying data in properties and fields. You should declare internal details of a class as Private to prevent them from being used outside of your class; this technique is called data hiding, and is how customer information such as the account balance is protected.
Encapsulation - Cont.

One basic rule of encapsulation is that class data should be modified or retrieved only via Property procedures or methods. Hiding the implementation details of your classes prevents classes from being used in undesired ways, and lets you to later modify such items without risk of compatibility problems. For example, later versions of the BankAccount class could change the data type of the AccountBalance field without breaking other applications that rely on this field having a specific data type.
Inheritance

Inheritance describes the ability to create new classes based on an existing class. The new class inherits all the properties and methods and events of the base class, and can be customized with additional properties and methods. For example, you can create a new class named "Truck" based on the "Car" class. The "Truck" class inherits the "Color" property from the "Car" class and can have additional properties such as "FourWheelDrive."
Inheritance - Cont.

Like Visual Basic structures, you can use classes to define data types that encapsulate a group of related items. Unlike structures, however, Visual Basic classes can inherit and extend the characteristics of other classes. Classes that serve as a basis for new classes are called base classes. Classes derived from base classes are called derived classes. Derived classes inherit all the fields, properties, methods, and events of the base class. This means you can develop and debug a class once, and then reuse it as the basis for other classes.
Examples:

The following example defines a base class that represents a generic bank account, and a specific class that inherits the properties of the base class but is customized to describe a checking account.
Class BankAccount
    Private AccountNumber As String
    Private AccountBalance As Decimal
    Private HoldOnAccount As Boolean = False
    Public Sub PostInterest()
        ' Add code to calculate the interest for this account.
    End Sub
    ReadOnly Property Balance() As Decimal
        Get
            ' Return the available balance.
            Return AccountBalance
        End Get
    End Property
End Class

Class CheckingAccount
    Inherits BankAccount
    Sub ProcessCheck()
        ' Add code to process a check drawn on this account.
    End Sub
End Class
Polymorphism means that you can have multiple classes that can be used interchangeably, even though each class implements the same properties or methods in different ways. Polymorphism is important to object-oriented programming because it lets you use items with the same names, regardless of what type of object is in use at the moment.
For example, given a base class of "Car," polymorphism enables the programmer to define different "StartEngine" methods for any number of derived classes. The "StartEngine" method of a derived class named "DieselCar" may be completely different from the method with the same name in the base class. Other procedures or methods can use the "StartEngine" method of the derived classes in the same way, no matter what type of "Car" object is being used at the time.
Reference

http://msdn.microsoft.com/en-us/library/wawheh00%28v=vs.90%29.aspx