DOWNLOAD PDF OBJECTS HAVE CLASS

Chapter 1 : Object Class (System) | Microsoft Docs

Software objects are conceptually similar to real-world objects: they too consist of state and related behavior. The fact that the values (state) for name, color, breed, and hungry are stored in the object in attributes is an implementation detail.

Simplest programming tutorials for beginners What do you want to learn today? Meaning, it supports different programming styles. One of the popular ways to solve a programming problem is by creating objects, known as object-oriented style of programming. Object is simply a collection of data and functions that act on those data. A class is a blueprint for the object. We can think of class as a sketch prototype of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object. As, many houses can be made from the same description, we can create many objects from a class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end. This class has two data members: The private keyword makes data and functions private. Private data and functions can be accessed only from inside the same class. The public keyword makes data and functions public. Public data and functions can be accessed out of the class. Here, data1 and data2 are private members where as function1 and function2 are public members. If you try to access private data from outside of the class, compiler throws error. This feature in OOP is known as data hiding. To use the data and access functions defined in the class, you need to create objects. In the above class Test, data1 and data2 are data members and function1 and function2 are member functions. You can access the data members and member functions by using a. Similarly, the data member can be accessed as: So, you can use o2. However, the code o1. Two objects o1 and o2 of the same class are declared. The insertIntegerData function is called for the o1 object using: Then, the insertFloatData function for object o2 is called and the return value from the function is stored in variable secondDataOfObject2 using: You should also check these topics to learn more on objects and classes:

Chapter 2 : Object-oriented programming concepts: Objects and classes | Adobe Developer Connection

Examples. The following example defines a Point type derived from the Object class and overrides many of the virtual methods of the Object class. In addition, the example shows how to call many of the static and instance methods of the Object class.

Distributed object The object-oriented approach is not just a programming model. It can be used equally well as an interface definition language for distributed systems. The objects in a distributed computing model tend to be larger grained, longer lasting, and more service-oriented than programming objects. An IDL shields the client of all of the details of the distributed server object. Details such as which computer the object resides on, what programming language it uses, what operating system , and other platform specific issues. The IDL is also usually part of a distributed environment that provides services such as transactions and persistence to all objects in a uniform manner. Protocol objects are components of a protocol stack that enclose network communication within an object-oriented interface. Replicated objects are groups of distributed objects called replicas that run a distributed multi-party protocol to achieve high consistency between their internal states, and that respond to requests in a coordinated way. Live distributed objects or simply live objects [9] generalize the replicated object concept to groups of replicas that might internally use any distributed protocol, perhaps resulting in only a weak consistency between their local states. Some of these extensions, such as distributed objects and protocol objects, are domain-specific terms for special types of "ordinary" objects used in a certain context such as remote method invocation or protocol composition. Others, such as replicated objects and live distributed objects, are more non-standard, in that they abandon the usual case that an object resides in a single location at a time, and apply the concept to groups of entities replicas that might span across multiple locations, might have only weakly consistent state, and whose membership might dynamically change. RDF provides the capability to define basic objectsâ€”names, properties, attributes, relationsâ€”that are accessible via the Internet. OWL adds a richer object model, based on set theory, that provides additional modeling capabilities such as multiple inheritance. OWL objects are not like standard large grained distributed objects accessed via an Interface Definition Language. Such an approach would not be appropriate for the Internet because the Internet is constantly evolving and standardization on one set of interfaces is difficult to achieve. However, there are important distinctions between OWL objects and traditional object-oriented programming objects. Where as traditional objects get compiled into static hierarchies usually with single inheritance, OWL objects are dynamic. An OWL object can change its structure at run time and can become an instance of new or different classes. Another critical difference is the way the model treats information that is currently not in the system. Programming objects and most database systems use the " closed-world assumption ". If a fact is not known to the system that fact is assumed to be false. Semantic Web objects use the open-world assumption , a statement is only considered false if there is actual relevant information that it is false, otherwise it is assumed to be unknown, neither true nor false. Classes are regarded as sets of individuals. Instances can not change their type at runtime. Class membership may change at runtime. The list of classes is fully known at compile-time and cannot change after that. Classes can be created and changed at runtime. Compilers are used at build-time. Compile-time errors indicate problems. Reasoners can be used for classification and consistency checking at runtime or build-time. Classes encode much of their meaning and behavior through imperative functions and methods. Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached. Instances are anonymous insofar that they cannot easily be addressed from outside of an executing program. If there is not enough information to prove a statement true, then it is assumed to be false. If there is not enough information to prove a statement true, then it may be true or false.

Chapter 3 : what is the Class object (calendrierdelascience.com)? - Stack Overflow

In C++, all objects have a type. All the word "object" means is "an instance of a type". Let's look at a few examples: int foo; In this case, foo is an object who's type is int.

It was developed as a way to handle the rapidly increasing size and complexity of software systems, and to make it easier to modify these large and complex systems over time. Up to now, most of the programs we have been writing use a procedural programming paradigm. In procedural programming the focus is on writing functions or procedures which operate on data. In object-oriented programming the focus is on the creation of objects which contain both data and functionality together. Usually, each object definition corresponds to some object or concept in the real world, and the functions that operate on that object correspond to the ways real-world objects interact. We are now ready to create our own user-defined class: Consider the concept of a mathematical point. In two dimensions, a point is two numbers coordinates that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, 0, 0 represents the origin, and x, y represents the point x units to the right and y units up from the origin. Some of the typical operations that one associates with points might be calculating the distance of a point from the origin, or from another point, or finding a midpoint of two points, or asking if a point falls within a given rectangle or circle. A natural way to represent a point in Python is with two numeric values. The question, then, is how to group these two values into a compound object. The quick and dirty solution is to use a tuple, and for some applications that might be a good choice. An alternative is to define a new class. This approach involves a bit more effort, but it has advantages that will be apparent soon. The syntax rules for a class definition are the same as for other compound statements. There is a header which begins with the keyword, class, followed by the name of the class, and ending with a colon. Indentation levels tell us where the class ends. If the first line after the class header is a string, it becomes the docstring of the class, and will be recognized by various tools. This is also the way docstrings work in functions. This initializer method is automatically called whenever a new instance of Point is created. The self parameter we could choose any other name, but self is the convention is automatically set to reference the newly created object that needs to be initialized. A function like Turtle or Point that creates a new object instance is called a constructor, and every class automatically provides a constructor function which is named the same as the class. It may be helpful to think of a class as a factory for making objects. As the object comes off the production line, its initialization method is executed to get the object properly set up with its factory default settings. We can modify the attributes in an instance using dot notation: In this case the attribute we are selecting is a data item from an instance. The following state diagram shows the result of these assignments: The variable p refers to a Point object, which contains two attributes. Each attribute refers to a number. We can access the value of an attribute using the same syntax: In this case, we assign that value to a variable named x. There is no conflict between the variable x in the global namespace here and the attribute x in the namespace belonging to the instance. The purpose of dot notation is to fully qualify which variable we are referring to unambiguously. We can use dot notation as part of any expression, so the following statements are legal: The second line calculates the value The x and y parameters here are both optional. Here is our improved class in action: We can add methods to the Point class that are sensible operations for points, but which may not be appropriate for other tuples like 25, 12 which might represent, say, a day and a month, e. So being able to calculate the distance from the origin is sensible for points, but not for day, month data. We can group together the sensible operations, and the kinds of data they apply to, and each instance of the class can have its own state. A method behaves like a function but it is invoked on a specific instance, e. Like a data attribute, methods are accessed using dot notation. We must run our program first, to make our Point class available to the interpreter. As already noted, it is customary to name this parameter self. Be aware that our variable only holds a reference to an object, so passing tess into a function creates an alias: Here is a simple function involving our new Point objects: A better approach is to have a method so that every instance can produce a string representation of itself. But these automatic mechanisms do not yet do exactly what we want:

For example, given two Point objects, find their midpoint. Suppose we have a point object, and wish to find the midpoint halfway between it and some other target point: Just as function calls are composable, method calls and object instantiation are also composable, leading to this alternative that uses no variables: This change in perspective might be more polite, but it may not initially be obvious that it is useful. But sometimes shifting responsibility from the functions onto the objects makes it possible to write more versatile functions, and makes it easier to maintain and reuse code. The most important advantage of the object-oriented style is that it fits our mental chunking and real-life experience more accurately. The functionality of real-world objects tends to be tightly bound up inside the objects themselves. OOP allows us to accurately mirror this when we organize our programs. Consider a turtle object. Its state consists of things like its position, its heading, its color, and its shape. For a bank account object, a main component of the state would be the current balance, and perhaps a log of all transactions. The methods would allow us to query the current balance, deposit new funds, or make a payment. Making a payment would include an amount, and a description, so that this could be added to the transaction log. A class can also be thought of as a template for the objects that are instances of it. The iPhone is a class. By December , estimates are that 50 million instances had been sold! If the class has an initializer method, this method is used to get the attributes i. Instance and object are used interchangeably. It bundles together the data and the operations that are relevant for that kind of data. For example, Point 3, 5. The coefficients a and b completely describe the line. Write a method in the Point class so that if a point instance is given another point, it will compute the equation of the straight line joining the two points. It must return the two coefficients as a tuple of two values. When will this method fail? Given four points that fall on the circumference of a circle, find the midpoint of the circle. When will this function fail? You must know how to solve the geometry problem before you think of going anywhere near programming. Each message will be represented as a tuple:

Chapter 4 : Object (Java Platform SE 7)

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts âˆ' Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have.

An Object Class is a part of the standard SCP template and serves as a rough indicator for how difficult an object is to contain. In universe, Object Classes are for the purposes of identifying containment needs, research priority, budgeting, and other considerations. Safe Safe-class SCPs are anomalies that are easily and safely contained. This is often due to the fact that the Foundation has researched the SCP well enough that containment does not require significant resources or that the anomalies require a specific and conscious activation or trigger. Classifying an SCP as Safe, however, does not mean that handling or activating it does not pose a threat. For a complete list of Safe-class articles on the site, click here. Usually this is because the SCP is insufficiently understood or inherently unpredictable. For a complete list of Euclid-class articles on the site, click here. Keter Keter-class SCPs are anomalies that are exceedingly difficult to contain consistently or reliably, with containment procedures often being extensive and complex. For a complete list of Keter-class articles on the site, click here. Even the mere existence of Thaumiel-class objects is classified at the highest levels of the Foundation and their locations, functions, and current status are known to few Foundation personnel outside of the O5 Council. For a complete list of Thaumiel-class articles on the site, click here. Neutralized Neutralized SCPs are anomalies that are no longer anomalous, either through having been intentionally or accidentally destroyed, or disabled. For a complete list of Neutralized-class articles on the site, click here. Explained Explained SCPs are commonly articles about anomalies that are completely and fully understood to the point where their effects are now explainable by mainstream science or phenomena that have been debunked or falsely mistaken as an anomaly. For a complete list of Explained-class articles on the site, click here. They are generally only used once and are created to further the narrative in a particular SCP. While some authors choose to introduce exceptions to these rules, they are only very rarely done and need to justify their existence and placement. Many site members will downvote for non-standard Object Classes if used without merit. For a complete list of articles with Esoteric classes on the site, click here. Decommissioned Decommissioned SCPs are an Object Class that was used by senior staff in the past to not only delete unwanted articles but place them in a sort of "Wall of Shame" to serve as examples of what not to do. This Object Class is not used anymore. Decommissioning articles is not done anymore, partly because such heavy-handed edits by SCP staff are no longer allowed and partly because decommissioning ended up backfiring. It goes like this: If you find an SCP article that you feel might be inappropriately classified, feel free to raise discussion on the topic and see what other community members think. If the explanation is not to your satisfaction, then feel free to express your opinion on the matter and vote accordingly on the page. If you have any other questions about Object Classes, feel free to ask in the discussion.

Chapter 5 : What are Objects, Methods, Classes, Properties in UFT/QTP ? - TestNBug

Class vs. type. In casual use, people often refer to the "class" of an object, but narrowly speaking objects have type: the interface, namely the types of member variables, the signatures of member functions (methods), and properties these satisfy.

Message Parsing In this chapter, we will look into the concepts - Classes and Objects. A dog has states - color, name, breed as well as behaviors â€” wagging the tail, barking, eating. An object is an instance of a class. Objects in Java Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior. If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running. If you compare the software object with a real-world object, they have very similar characteristics. Software objects also have a state and a behavior. So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods. Classes in Java A class is a blueprint from which individual objects are created. Following is a sample of a class. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class. A class can have any number of methods to access the value of various kinds of methods. In the above example, barking , hungry and sleeping are methods. Following are some of the important topics that need to be discussed when looking into classes of the Java Language. Constructors When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class. Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor. We are going to discuss constructors in detail in the subsequent chapters. Creating an Object As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects. This call initializes the new object. These rules are essential when declaring classes, import statements and package statements in a source file. There can be only one public class per source file. A source file can have multiple non-public classes. The public class name should be the name of the source file as well which should be appended by. If the class is defined inside a package, then the package statement should be the first statement in the source file. If import statements are present, then they must be written between the package statement and the class declaration. If there are no package statements, then the import statement should be the first line in the source file. Import and package statements will imply to all the classes present in the source file. Classes have several access levels and there are different types of classes; abstract classes, final classes, etc. We will be explaining about all these in the access modifiers chapter. Apart from the above mentioned types of classes, Java also has some special classes called Inner classes and Anonymous classes. Java Package In simple words, it is a way of categorizing the classes and interfaces. When developing applications in Java, hundreds of classes and interfaces will be written, therefore categorizing these classes is a must as well as makes life much easier. Import Statements In Java if a fully qualified name, which includes the package and the class name is given, then the compiler can easily locate the source code or classes. Import statement is a way of giving the proper location for the compiler to find that particular class. They are Employee and EmployeeTest. First open notepad and add the following code. Remember this is the Employee class and the class is a public class. Now, save this source file with the name Employee. The Employee class has four instance variables - name, age, designation and salary. The class has one explicitly defined constructor, which takes a parameter. Therefore, in order for us to run this Employee class there should be a main method and objects should be created. We will be creating a separate class for these tasks. Following is the EmployeeTest class, which creates two instances of the class Employee and invokes the methods for each object to assign values for each variable. Save the following code in EmployeeTest. Senior Software Engineer Salary: In the next session, we will discuss the basic data types in

Java and how they can be used when developing Java applications.

Chapter 6 : Class (computer programming) - Wikipedia

Similarly, for the 'WINBUTTON', which is the class and has objects 'Ok', 'Cancel', and 'Help'. The method that is applicable to all the three is 'Click'. It is not necessary that all the objects have same class, can be multiples as well.

In general, the toString method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method. In other words, this method returns a string equal to the value of: If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. The awakened thread will not be able to proceed until the current thread relinquishes the lock on this object. The awakened thread will compete in the usual manner with any other threads that might be actively competing to synchronize on this object; for example, the awakened thread enjoys no reliable privilege or disadvantage in being the next thread to lock this object. By executing a synchronized instance method of that object. By executing the body of a synchronized statement that synchronizes on the object. For objects of type Class, by executing a synchronized static method of that class. The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object. The awakened threads will compete in the usual manner with any other threads that might be actively competing to synchronize on this object; for example, the awakened threads enjoy no reliable privilege or disadvantage in being the next thread to lock this object. See the notify method for a description of the ways in which a thread can become the owner of a monitor. This method causes the current thread call it T to place itself in the wait set for this object and then to relinquish any and all synchronization claims on this object. Thread T becomes disabled for thread scheduling purposes and lies dormant until one of four things happens: Some other thread invokes the notify method for this object and thread T happens to be arbitrarily chosen as the thread to be awakened. Some other thread invokes the notifyAll method for this object. Some other thread interrupts thread T. The specified amount of real time has elapsed, more or less. If timeout is zero, however, then real time is not taken into consideration and the thread simply waits until notified. The thread T is then removed from the wait set for this object and re-enabled for thread scheduling. It then competes in the usual manner with other threads for the right to synchronize on the object; once it has gained control of the object, all its synchronization claims on the object are restored to the status quo ante - that is, to the situation as of the time that the wait method was invoked. Thread T then returns from the invocation of the wait method. Thus, on return from the wait method, the synchronization state of the object and of thread T is exactly as it was when the wait method was invoked. A thread can also wake up without being notified, interrupted, or timing out, a so-called spurious wakeup. While this will rarely occur in practice, applications must guard against it by testing for the condition that should have caused the thread to be awakened, and continuing to wait if the condition is not satisfied. In other words, waits should always occur in loops, like this one: If the current thread is interrupted by any thread before or while it is waiting, then an InterruptedException is thrown. This exception is not thrown until the lock status of this object has been restored as described above. Note that the wait method, as it places the current thread into the wait set for this object, unlocks only this object; any other objects on which the current thread may be synchronized remain locked while the thread waits. IllegalArgumentException - if the value of timeout is negative. InterruptedException - if any thread interrupted the current thread before or while the current thread was waiting for a notification. The interrupted status of the current thread is cleared when this exception is thrown. This method is similar to the wait method of one argument, but it allows finer control over the amount of time to wait for a notification before giving up. The amount of real time, measured in nanoseconds, is given by: In particular, wait 0, 0 means the same thing as wait 0. The thread releases ownership of this monitor and waits until either of the following two conditions has occurred: The timeout period, specified by timeout milliseconds plus nanos nanoseconds arguments, has elapsed. The thread then waits until it can re-obtain ownership of the monitor and resumes execution. As in the one argument version, interrupts and spurious wakeups are possible, and this method should always be used in a loop: IllegalArgumentException - if the value of timeout is negative or the value of nanos is not in the range

In other words, this method behaves exactly as if it simply performs the call wait 0.

Chapter 7 : C++ Classes and Objects

Because each class is designed to have only a single responsibility, many classes are used to build an entire application. An instance is a specific object built from a specific class. It is assigned to a reference variable that is used to access all of the instance's properties and methods.

There are some subtle aspects to a number of these methods, especially the clone method. The clone Method If a class, or one of its superclasses, implements the Cloneable interface, you can use the clone method to create a copy from an existing object. To create a clone, you write: If the object does not, the method throws a CloneNotSupportedException exception. Exception handling will be covered in a later lesson. For the moment, you need to know that clone must be declared as protected Object clone throws CloneNotSupportedException or: If, however, an object contains a reference to an external object, say ObjExternal, you may need to override clone to get correct behavior. Otherwise, a change in ObjExternal made by one object will be visible in its clone also. Then the original object references ObjExternal and the clone references a clone of ObjExternal, so that the object and its clone are truly independent. The equals Method The equals method compares two objects for equality and returns true if they are equal. For primitive data types, this gives the correct result. For objects, however, it does not. To test whether two objects are equal in the sense of equivalency containing the same information , you must override the equals method. Here is an example of a Book class that overrides equals: They are considered equal because the objects compared contain the same ISBN number. You should always override the equals method if the identity operator is not appropriate for your class. If you override equals , you must override hashCode as well. The finalize Method The Object class provides a callback method, finalize , that may be invoked on an object when it becomes garbage. The finalize method may be called automatically by the system, but when it is called, or even if it is called, is uncertain. Therefore, you should not rely on this method to do your cleanup for you. The getClass Method You cannot override getClass. The getClass method returns a Class object, which has methods you can use to get information about the class, such as its name getSimpleName , its superclass getSuperclass , and the interfaces it implements getInterfaces. For example, the following method gets and displays the class name of an object: For example, you can test to see if the class is an annotation isAnnotation , an interface isInterface , or an enumeration isEnum. By definition, if two objects are equal, their hash code must also be equal. Therefore, if you override the equals method, you must also override the hashCode method as well. The toString Method You should always consider overriding the toString method in your classes. The String representation for an object depends entirely on the object, which is why you need to override toString in your classes. You can use toString along with System.

Chapter 8 : How many object can a class have? (Beginning Java forum at Coderanch)

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types.

An object can be a piece of an application, like a control or a form. An entire application can also be an object. When you create an application in Visual Basic, you constantly work with objects. You can use objects provided by Visual Basic, such as controls, forms, and data access objects. You can also use objects from other applications within your Visual Basic application. You can even create your own objects and define additional properties and methods for them. Objects act like prefabricated building blocks for programs â€" they let you write a piece of code once and reuse it over and over. This topic discusses objects in detail. Objects and classes Each object in Visual Basic is defined by a class. A class describes the variables, properties, procedures, and events of an object. Objects are instances of classes; you can create as many objects you need once you have defined a class. To understand the relationship between an object and its class, think of cookie cutters and cookies. The cookie cutter is the class. It defines the characteristics of each cookie, for example size and shape. The class is used to create objects. The objects are the cookies. You must create an object before you can access its members. To create an object from a class Determine from which class you want to create an object. Write a Dim Statement to create a variable to which you can assign a class instance. The variable should be of the type of the desired class. Dim nextCustomer As customer Add the New Operator keyword to initialize the variable to a new instance of the class. Dim nextCustomer As New customer You can now access the members of the class through the object variable. This is called early binding. For more information, see Object Variable Declaration. Multiple instances Objects newly created from a class are often identical to each other. Once they exist as individual objects, however, their variables and properties can be changed independently of the other instances. For example, if you add three check boxes to a form, each check box object is an instance of the CheckBox class. The individual CheckBox objects share a common set of characteristics and capabilities properties, variables, procedures, and events defined by the class. However, each has its own name, can be separately enabled and disabled, and can be placed in a different location on the form. Object members An object is an element of an application, representing an instance of a class. Fields, properties, methods, and events are the building blocks of objects and constitute their members. Member Access You access a member of an object by specifying, in order, the name of the object variable, a period. The following example sets the Text property of a Label object. If you type the period following the name of a variable declared as an instance of that class, IntelliSense lists all the instance members and none of the shared members. If you type the period following the class name itself, IntelliSense lists all the shared members and none of the instance members. For more information, see Using IntelliSense. Fields and properties Fields and properties represent information stored in an object. You retrieve and set their values with assignment statements the same way you retrieve and set local variables in a procedure. The following example retrieves the Width property and sets the ForeColor property of a Label object. Red Note that a field is also called a member variable. Use property procedures when: You need to control when and how a value is set or retrieved. The property has a well-defined set of values that need to be validated. Setting the property causes changes to other internal variables or to the values of other properties. A set of steps must be performed before the property can be set or retrieved. The value is of a self-validating type. For example, an error or automatic data conversion occurs if a value other than True or False is assigned to a Boolean variable. Any value in the range supported by the data type is valid. This is true of many properties of type Single or Double. The property is a String data type, and there is no constraint on the size or value of the string. For more information, see Property Procedures. Methods A method is an action that an object can perform. For example, Add is a method of the ComboBox object that adds a new entry to a combo box. The following example demonstrates the Start method of a Timer object. Dim safetyTimer As New System. Start Note that a method is simply a procedure that is exposed by an object. For more information, see Procedures. Events An event is an action recognized by an object, such as clicking the mouse or pressing a key, and for which you can write code to respond. Events can occur as a result of a user action or program code, or they can be caused by the system. Code that signals an event is said to raise the event, and code that responds to it is said to handle it. You can also develop your own custom events to be raised by your objects and handled by other objects. For more information, see Events. Instance members and shared members When you create an object from a class, the result is an instance of that class. Members that are not declared with the Shared keyword are instance members, which belong strictly to that particular instance. An instance member in one instance is independent of the same member in another instance of the same class. An instance member variable, for example, can have different values in different instances. Members declared with the Shared keyword are shared members, which belong to the class as a whole and not to any particular instance. A shared member exists only once, no matter how many instances of its class you create, or even if you create no instances. A shared member variable, for example, has only one value, which is available to all code that can access the class. Accessing nonshared members To access a nonshared member of an object Make sure the object has been created from its class and assigned to an object variable. Dim secondForm As New System. Form In the statement that accesses the member, follow the object variable name with the member-access operator. Show Accessing shared members To access a shared member of an object Follow the class name with the member-access operator. You should always access a Shared member of the object directly through the class name. Differences between classes and modules The main difference between classes and modules is that classes can be instantiated as objects while standard modules cannot. In contrast, object data exists separately for each instantiated object. Another difference is that unlike standard modules, classes can implement interfaces. Note When the Shared modifier is applied to a class member, it is associated with the class itself instead of a particular instance of the class. The member is accessed directly by using the class name, the same way module members are accessed. Classes and modules also use different scopes for their members. Members defined within a class are scoped within a specific instance of the class and exist only for the lifetime of the object. To access class members from outside a class, you must use fully qualified names in the format of Object. On the other hand, members declared within a module are publicly accessible by default, and can be accessed by any code that can access the module. This means that variables in a standard module are effectively global variables because they are visible from anywhere in your project, and they exist for the life of the program. Reusing classes and objects Objects let you declare variables and procedures once and then reuse them whenever needed. For example, if you want to add a spelling checker to an application you could define all the variables and support functions to provide spell-checking functionality. If you create your spelling checker as a class, you can then reuse it in other applications by adding a reference to the compiled assembly. Better yet, you may be able to save yourself some work by using a spelling checker class that someone else has already developed. NET Framework provides many examples of components that are available for use. The following example uses the TimeZone class in the System namespace. TimeZone provides members that allow you to retrieve information about the time zone of the current computer system. Relationships among objects Objects can be related to each other in several ways. The principal kinds of relationship are hierarchical and containment. Hierarchical relationship When classes are derived from more fundamental classes, they are said to have a hierarchical relationship. Class hierarchies are useful when describing items that are a subtype of a more general class. In the following example, suppose you want to define a special kind of Button that acts like a normal Button but also exposes a method that reverses the foreground and background colors. To define a class is derived from an already existing class Use a Class Statement to define a class from which to create the object you need. Public Class reversibleButton Be sure an End Class statement follows the last line of code in your class. By default, the integrated development environment IDE automatically generates an End Class when you enter a Class statement.

Chapter 9 : Object (computer science) - Wikipedia

Understanding classes and objects In object-oriented terminology, a class is a template for defining objects. It specifies the names and types of variables that can exist in an object, as well as "methods"--procedures for operating on those variables.

In addition, the example shows how to call many of the static and instance methods of the Object class. GetType If Not objType. Because all classes in the. NET Framework are derived from Object , every method defined in the Object class is available in all objects in the system. Derived classes can and do override some of these methods, including: Equals - Supports comparisons between objects. Finalize - Performs cleanup operations before an object is automatically reclaimed. GetHashCode - Generates a number corresponding to the value of the object to support the use of a hash table. ToString - Manufactures a human-readable text string that describes an instance of the class. Performance Considerations If you are designing a class, such as a collection, that must handle any type of object, you can create class members that accept instances of the Object class. However, the process of boxing and unboxing a type carries a performance cost. If you know your new class will frequently handle certain value types you can use one of two tactics to minimize the cost of boxing. Create a general method that accepts an Object type, and a set of type-specific method overloads that accept each value type you expect your class to frequently handle. If a type-specific method exists that accepts the calling parameter type, no boxing occurs and the type-specific method is invoked. If there is no method argument that matches the calling parameter type, the parameter is boxed and the general method is invoked. Design your type and its members to use generics. The common language runtime creates a closed generic type when you create an instance of your class and specify a generic type argument. The generic method is type-specific and can be invoked without boxing the calling parameter. Although it is sometimes necessary to develop general purpose classes that accept and return Object types, you can improve performance by also providing a type-specific class to handle a frequently used type. For example, providing a class that is specific to setting and getting Boolean values eliminates the cost of boxing and unboxing Boolean values. Constructors Initializes a new instance of the Object class.