

Chapter
1

CLASS

*Get on the
Fast Track!*



TM

**SYS-ED/
COMPUTER
EDUCATION
TECHNIQUES, INC.**

1 Structured Types

A structured type holds more than one value. The components of structured types can be manipulated individually or as a whole, and can themselves be structured types.

A structured type can have nested levels. There is no limit to the number of such nested structures.

Class types and class reference types are the cornerstones of object oriented programming in Object Pascal.

The reserved word `packed` in a structured type's declaration tells the compiler to compress data storage, even at the cost of diminished access to a component of a variable of this type.

Delphi's structured types are:

Type	Purpose
Array	<p>Arrays are one dimensional or multidimensional containers that hold multiple variables of the same type.</p> <p>Each variable of the array can be referred to by the array name and an index which is enclosed in brackets.</p>
File	<p>A file type is a linear sequence of elements that can be of any type except the following:</p> <ul style="list-style-type: none"> C File type C Any structured type with a file-type component C An object
Class	A class type is a structure consisting of a fixed number components.
Class Reference	Class-reference types allow operations to be performed directly on classes. This contrasts with class types, which allow operations to be performed on instances of classes.
Record	A record type is a collection of fields that can be of different types.
Set	A set type is a collection of objects of the same ordinal type. To declare a set type use the reserved words <code>set of</code> followed by the base type.

2 Class Types

A class type is a structure consisting of a fixed number components.

Possible components of a class are:

- C Fields.
- C Methods.
- C Properties.

Unlike other types, a class type can be declared only in a type declaration part in the outermost scope of a program or unit. Therefore, a class type cannot be declared in a variable declaration part or within a procedure, function, or method block.

A class type is declared using the reserved word `class` and defines the contents of a class. Classes are also called "object types." The two terms are interchangeable.

A class type must be globally declared. A class type cannot be declared in a variable declaration part or within a procedure, function, or method block.

2.1 Inheritance

A class type can inherit components from another class type. The inheriting class is a descendant and the class inherited from is its ancestor.

Inheritance is transitive; if T3 inherits from T2, and T2 inherits from T1, then T3 also inherits from T1. The domain of a class type consists of itself and all its descendants.

A descendant class implicitly contains all the components defined by its ancestor classes. A descendant class can add new components to those it inherits. However, it cannot remove the definition of a component defined in an ancestor class.

The predefined class type `TObject` is the ultimate ancestor of all class types. If the declaration of a class type does not specify an ancestor type (that is, if the heritage part of the class declaration is omitted), the class type will be derived from `TObject`.

`TObject` is declared by the `System` unit, and defines a number of methods that apply to all classes.

2.2 Class-type Compatibility

A class type is assignment-compatible with any ancestor object type; therefore, during program execution, a class type variable can reference an instance of that type or an instance of any descendant type.

Example:

Given the declarations:

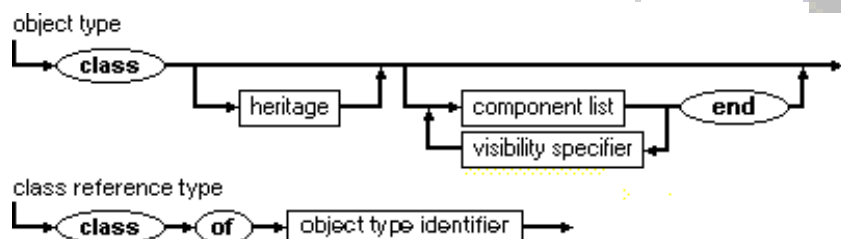
```
type
  Tfigure = class
  :
  end;
  Trectangle = class(Tfigure)
  :
  end;
  TroundRect = class(Trectangle)
  :
  end;
  Tellipse = class(Tfigure)
  :
  end;
```

A value of type TRectangle can be assigned to variables of type TRectangle, TFigure, and TObject.

During execution of a program, a variable of type TFigure might be either nil or reference an instance of TFigure, TRectangle, TRoundRect, TELLipse, or any other instance of a descendant of TFigure.

3 Declaring an Object Type

The class reserved word is used to declare an object type or class method. It is also used to define a class reference type.



The declaration of a field specifies an identifier that names the field, and its data type.

The declaration of a method specifies a procedure, function, constructor, or destructor heading.

The definition of a property names the property and its access methods, and may provide information on how the property behaves during streaming.

An object type can inherit components from another object type. The inheriting object is a descendant and the object inherited from is an ancestor.

The domain of an object type consists of itself and all its descendants.

A class reference type is defined using the sequence of reserved words `class of` followed by the name of a class. A variable of a class reference type can be set at run time to refer either to the class named in the declaration or any of its subclasses.

4 Methods

A method is a procedure or function declared inside an object-type declaration that performs an operation on an object.

Methods can access the object's data fields without having to pass parameters.

The declaration inside the object-type declaration corresponds to a forward declaration of that method.

The body of the method is defined outside the object type declaration, but within the same scope. Its header must contain the name of the object type it is bound to such as:

```
procedure ObjectType.Method(Param1, Param2: Integer);  
begin  
  ...  
end; (* Method *)
```

Within the implementation of a method, the identifier `Self` represents an implicit parameter that references the object for which the method was invoked.

Methods declared in an object type are by default static. When a static method is called, the declared (compile-time) type of the class or object used in the method call determines which method implementation to activate.

The types of the class or object used in the method call are:

- C Virtual methods.
- C Dynamic methods.
- C Class methods.
- C Override directive.
- C Message handling methods.
- C Abstract methods.
- C Constructors and destructors are special methods that control construction and destruction of objects.

Within a method, a function call or procedure statement allows a qualified method designator to activate a specific method. This type of call is known as a qualified method activation.

4.1 Virtual Methods

Virtual methods are resolved by the compiler at run-time; this process is known as late binding. By default, all methods are static except constructor methods. However, any methods can be virtualized by including a virtual directive in the method declaration.

When a virtual method is called, the actual (run-time) type of the class or object used in the method call determines which method implementation to activate.

4.2 Overriding a Virtual Method

An object type can override any of the methods it inherits from its ancestors. The scope of an override method extends over all of the descendants of the defining object, or until the method identifier is redefined.

An override of a virtual method must match exactly the order, types, and names of the parameters, and the type of the function result, if any. The override must include the override directive in place of the virtual method.

The only way to override a virtual method is through the override directive. If a method declaration in a descendant class specifies the same method identifier as an inherited method, but does not specify an override directive, the new method declaration will hide the inherited declaration, but not override it.

Example:

The following two descendant classes override the Draw method inherited from TFigure.

```
type
  Trectangle = class(TFigure)
    procedure Draw; override;
  end;
  Tellipse = class(TFigure)
    procedure Draw; override;
  end;
```

Example:

The following section of code illustrates the effect of calling a virtual method through a class type variable whose actual type varies at run-time.

```
var
  Figure: TFigure;
begin
  Figure := TRectangle.Create;           { Invokes TRectangle.Draw }
  Figure.Draw;
  Figure.Destroy;
  Figure := TEllipse.Create;           { Invokes TEllipse.Draw }
  Figure.Draw;
  Figure.Destroy;
end;
```