PASCAL - CLASSES

You have seen that Pascal Objects exhibit some characteristics of object-oriented paradigm. They implement encapsulation, data hiding and inheritance, but they also have limitations. For example, Pascal Objects do not take part in polymorphism. So classes are widely used to implement proper object-oriented behavior in a program, especially the GUI-based software.

A Class is defined in almost the same way as an Object, but is a pointer to an Object rather than the Object itself. Technically, this means that the Class is allocated on the Heap of a program, whereas the Object is allocated on the Stack. In other words, when you declare a variable the object type, it will take up as much space on the stack as the size of the object, but when you declare a variable of the class type, it will always take the size of a pointer on the stack. The actual class data will be on the heap.

Defining Pascal Classes

A class is declared in the same way as an object, using the type declaration. The general form of a class declaration is as follows −

```
type class-identifier = class
   private
      field1 : field-type;
      field2 : field-type;
        ...

   public
      constructor create();
      procedure proc1;
      function f1(): function-type;
end;
var classvar : class-identifier;
```

Its worth to note following important points −

- Class definitions should come under the type declaration part of the program only.
- A class is defined using the **class** keyword.
- Fields are data items that exist in each instance of the class.
- Methods are declared within the definition of a class.
- There is a predefined constructor called **Create** in the Root class. Every abstract class and every concrete class is a descendant of Root, so all classes have at least one constructor.
- There is a predefined destructor called **Destroy** in the Root class. Every abstract class and every concrete class is a descendant of Root, so, all classes have at least one destructor.

Let us define a Rectangle class that has two integer type data members - length and width and some member functions to manipulate these data members and a procedure to draw the rectangle.

```
type
   Rectangle = class
   private
      length, width: integer;

   public
      constructor create(l, w: integer);
      procedure setlength(l: integer);
      function getlength(): integer;
      procedure setwidth(w: integer);
      function getwidth(): integer;
```

```
      procedure draw;
end;
```

Let us write a complete program that would create an instance of a rectangle class and draw the rectangle. This is the same example we used while discussing Pascal Objects. You will find both programs are almost same, with the following exceptions −

- You will need to include the {$mode objfpc} directive for using the classes.
- You will need to include the {$m+} directive for using constructors.
- Class instantiation is different than object instantiation. Only declaring the variable does not create space for the instance, you will use the constructor create to allocate memory.

Here is the complete example −

```
{$mode objfpc} // directive to be used for defining classes
{$m+}     // directive to be used for using constructor

program exClass;
type
   Rectangle = class
   private
      length, width: integer;

   public
      constructor create(l, w: integer);
      procedure setlength(l: integer);

      function getlength(): integer;
      procedure setwidth(w: integer);

      function getwidth(): integer;
      procedure draw;
end;
var
   r1: Rectangle;

constructor Rectangle.create(l, w: integer);
begin
   length := l;
   width := w;
end;

procedure Rectangle.setlength(l: integer);
begin
   length := l;
end;

procedure Rectangle.setwidth(w: integer);
begin
   width :=w;
end;

function Rectangle.getlength(): integer;
begin
   getlength := length;
end;

function Rectangle.getwidth(): integer;
begin
   getwidth := width;
end;

procedure Rectangle.draw;
var
   i, j: integer;
begin
```

```
   for i:= 1 to length do
   begin
      for j:= 1 to width do
         write(' * ');
      writeln;
   end;
end;

begin
   r1:= Rectangle.create(3, 7);

   writeln(' Draw Rectangle: ', r1.getlength(), ' by ' , r1.getwidth());
   r1.draw;
   r1.setlength(4);
   r1.setwidth(6);

   writeln(' Draw Rectangle: ', r1.getlength(), ' by ' , r1.getwidth());
   r1.draw;
end.
```

When the above code is compiled and executed, it produces the following result −

```
Draw Rectangle: 3 by 7
* * * * * * *
* * * * * * *
* * * * * * *
Draw Rectangle: 4 by 6
* * * * * *
* * * * * *
* * * * * *
* * * * * *
```

Visibility of the Class Members

Visibility indicates the accessibility of the class members. Pascal class members have five types of visibility −

Visibility	Accessibility
Public	These members are always accessible.
Private	These members can only be accessed in the module or unit that contains the class definition. They can be accessed from inside the class methods or from outside them.
Strict Private	These members can only be accessed from methods of the class itself. Other classes or descendent classes in the same unit cannot access them.
Protected	This is same as private, except, these members are accessible to descendent types, even if they are implemented in other modules.
Published	This is same as a Public, but the compiler generates type information that is needed for automatic streaming of these classes if the compiler is in the {$M+} state. Fields defined in a published section must be of class type.

Constructors and Destructors for Pascal Classes

Constructors are special methods, which are called automatically whenever an object is created. So we take full advantage of this behavior by initializing many things through constructor functions.

Pascal provides a special function called create to define a constructor. You can pass as many

arguments as you like into the constructor function.

Following example will create one constructor for a class named Books and it will initialize price and title for the book at the time of object creation.

```
program classExample;

{$MODE OBJFPC} //directive to be used for creating classes
{$M+} //directive that allows class constructors and destructors
type
   Books = Class
   private
      title : String;
      price: real;

   public
      constructor Create(t : String; p: real); //default constructor

      procedure setTitle(t : String); //sets title for a book
      function getTitle() : String; //retrieves title

      procedure setPrice(p : real); //sets price for a book
      function getPrice() : real; //retrieves price

      procedure Display(); // display details of a book
end;
var
   physics, chemistry, maths: Books;

//default constructor
constructor Books.Create(t : String; p: real);
begin
   title := t;
   price := p;
end;

procedure Books.setTitle(t : String); //sets title for a book
begin
   title := t;
end;

function Books.getTitle() : String; //retrieves title
begin
   getTitle := title;
end;

procedure Books.setPrice(p : real); //sets price for a book
begin
   price := p;
end;

function Books.getPrice() : real; //retrieves price
begin
   getPrice:= price;
end;

procedure Books.Display();
begin
   writeln('Title: ', title);
   writeln('Price: ', price:5:2);
end;

begin
   physics := Books.Create('Physics for High School', 10);
   chemistry := Books.Create('Advanced Chemistry', 15);
   maths := Books.Create('Algebra', 7);

   physics.Display;
   chemistry.Display;
```

```
   maths.Display;
end.
```

When the above code is compiled and executed, it produces the following result −

```
Title: Physics for High School
Price: 10
Title: Advanced Chemistry
Price: 15
Title: Algebra
Price: 7
```

Like the implicit constructor named create, there is also an implicit destructor method destroy using which you can release all the resources used in the class.

Inheritance

Pascal class definitions can optionally inherit from a parent class definition. The syntax is as follows −

```
type
childClas-identifier = class(baseClass-identifier)
< members >
end;
```

Following example provides a novels class, which inherits the Books class and adds more functionality based on the requirement.

```
program inheritanceExample;

{$MODE OBJFPC} //directive to be used for creating classes
{$M+} //directive that allows class constructors and destructors

type
   Books = Class
   protected
      title : String;
      price: real;

   public
      constructor Create(t : String; p: real); //default constructor

      procedure setTitle(t : String); //sets title for a book
      function getTitle() : String; //retrieves title

      procedure setPrice(p : real); //sets price for a book
      function getPrice() : real; //retrieves price

      procedure Display(); virtual; // display details of a book
end;
(* Creating a derived class *)

type
   Novels = Class(Books)
   private
      author: String;

   public
      constructor Create(t: String); overload;
      constructor Create(a: String; t: String; p: real); overload;

      procedure setAuthor(a: String); // sets author for a book
      function getAuthor(): String; // retrieves author name

      procedure Display(); override;
end;
var
```

```
   n1, n2: Novels;

//default constructor
constructor Books.Create(t : String; p: real);
begin
   title := t;
   price := p;
end;

procedure Books.setTitle(t : String); //sets title for a book
begin
   title := t;
end;

function Books.getTitle() : String; //retrieves title
begin
   getTitle := title;
end;

procedure Books.setPrice(p : real); //sets price for a book
begin
   price := p;
end;

function Books.getPrice() : real; //retrieves price
begin
   getPrice:= price;
end;

procedure Books.Display();
begin
   writeln('Title: ', title);
   writeln('Price: ', price);
end;

(* Now the derived class methods  *)
constructor Novels.Create(t: String);
begin
   inherited Create(t, 0.0);
   author:= ' ';
end;

constructor Novels.Create(a: String; t: String; p: real);
begin
   inherited Create(t, p);
   author:= a;
end;

procedure Novels.setAuthor(a : String); //sets author for a book
begin
   author := a;
end;

function Novels.getAuthor() : String; //retrieves author
begin
   getAuthor := author;
end;

procedure Novels.Display();
begin
   writeln('Title: ', title);
   writeln('Price: ', price:5:2);
   writeln('Author: ', author);
end;

begin
   n1 := Novels.Create('Gone with the Wind');
   n2 := Novels.Create('Ayn Rand','Atlas Shrugged', 467.75);
   n1.setAuthor('Margaret Mitchell');
```

```
   n1.setPrice(375.99);
   n1.Display;
   n2.Display;
end.
```

When the above code is compiled and executed, it produces the following result −

```
Title: Gone with the Wind
Price: 375.99
Author: Margaret Mitchell
Title: Atlas Shrugged
Price: 467.75
Author: Ayn Rand
```

Its worth to note following important points −

- The members of the Books class have **protected** visibility.
- The Novels class has two constructors, so the **overload** operator is used for function overloading.
- The Books.Display procedure has been declared **virtual**, so that the same method from the Novels class can **override** it.
- The Novels.Create constructor calls the base class constructor using the **inherited** keyword.

Interfaces

Interfaces are defined to provide a common function name to the implementers. Different implementers can implement those interfaces according to their requirements. You can say, interfaces are skeletons, which are implemented by developers. Following is an example of interface −

```
type
   Mail = Interface
      Procedure SendMail;
      Procedure GetMail;
   end;

   Report = Class(TInterfacedObject,  Mail)
      Procedure SendMail;
      Procedure GetMail;
   end;
```

Please note that, when a class implements an interface, it should implement all methods of the interface. If a method of an interface is not implemented, then the compiler will give an error.

Abstract Classes

An abstract class is one that cannot be instantiated, only inherited. An abstract class is specified by including the word symbol abstract in the class definition, like this −

```
type
   Shape = ABSTRACT CLASS (Root)
      Procedure draw; ABSTRACT;
      ...
   end;
```

When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.

Static Keyword

Declaring class members or methods as static makes them accessible without needing an

instantiation of the class. A member declared as static cannot be accessed with an instantiated class object *thoughastaticmethodcan*. The following example illustrates the concept −

```
{$mode objfpc}
{$static on}
type
   myclass=class
      num : integer;static;
   end;
var
   n1, n2 : myclass;
begin
   n1:= myclass.create;
   n2:= myclass.create;
   n1.num := 12;
   writeln(n2.num);
   n2.num := 31;
   writeln(n1.num);
   writeln(myclass.num);
   myclass.num := myclass.num + 20;
   writeln(n1.num);
   writeln(n2.num);
end.
```

When the above code is compiled and executed, it produces the following result −

```
12
31
31
51
51
```

You must use the directive {$static on} for using the static members.