PASCAL - OBJECT ORIENTED

We can imagine our universe made of different objects like sun, earth, moon, etc. Similarly, we can imagine our car made of different objects like wheel, steering, gear, etc. Same way, there are object-oriented programming concepts, which assume everything as an object and implement a software using different objects. In Pascal, there are two structural data types used to implement a real world object −

- Object types
- Class types

Object-Oriented Concepts

Before we go in detail, let's define important Pascal terms related to Object-Oriented Pascal.

- **Object** − An Object is a special kind of record that contains fields like a record; however, unlike records, objects contain procedures and functions as part of the object. These procedures and functions are held as pointers to the methods associated with the object's type.
- **Class** − A Class is defined in almost the same way as an Object, but there is a difference in way they are created. The Class is allocated on the Heap of a program, whereas the Object is allocated on the Stack. It is a pointer to the object, not the object itself.
- **Instantiation of a class** − Instantiation means creating a variable of that class type. Since a class is just a pointer, when a variable of a class type is declared, there is memory allocated only for the pointer, not for the entire object. Only when it is instantiated using one of its constructors, memory is allocated for the object. Instances of a class are also called 'objects', but do not confuse them with Object Pascal Objects. In this tutorial, we will write 'Object' for Pascal Objects and 'object' for the conceptual object or class instance.
- **Member Variables** − These are the variables defined inside a Class or an Object.
- **Member Functions** − These are the functions or procedures defined inside a Class or an Object and are used to access object data.
- **Visibility of Members** − The members of an Object or Class are also called the fields. These fields have different visibilities. Visibility refers to accessibility of the members, i.e., exactly where these members will be accessible. Objects have three visibility levels: public, private and protected. Classes have five visibility types: public, private, strictly private, protected and published. We will discuss visibility in details.
- **Inheritance** − When a Class is defined by inheriting existing functionalities of a parent Class, then it is said to be inherited. Here child class will inherit all or few member functions and variables of a parent class. Objects can also be inherited.
- **Parent Class** − A Class that is inherited by another Class. This is also called a base class or super class.
- **Child Class** − A class that inherits from another class. This is also called a subclass or derived class.
- **Polymorphism** − This is an object-oriented concept where same function can be used for different purposes. For example, function name will remain same but it may take different number of arguments and can do different tasks. Pascal classes implement polymorphism. Objects do not implement polymorphism.
- **Overloading** − It is a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation. Pascal classes implement overloading, but the Objects do not.
- **Data Abstraction** − Any representation of data in which the implementation details are

hidden *abstracted*.

- **Encapsulation** − Refers to a concept where we encapsulate all the data and member functions together to form an object.
- **Constructor** − Refers to a special type of function which will be called automatically whenever there is an object formation from a class or an Object.
- **Destructor** − Refers to a special type of function which will be called automatically whenever an Object or Class is deleted or goes out of scope.

Defining Pascal Objects

An object is declared using the type declaration. The general form of an object declaration is as follows −

```
type object-identifier = object
   private
   field1 : field-type;
   field2 : field-type;
   ...
   public
   procedure proc1;
   function f1(): function-type;
   end;
var objectvar : object-identifier;
```

Let us define a Rectangle Object 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 = object
   private
      length, width: integer;

   public
      constructor init;
      destructor done;

      procedure setlength(l: inteter);
      function getlength(): integer;

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

      procedure draw;
end;
var
   r1: Rectangle;
   pr1: ^Rectangle;
```

After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.

Following example shows how to set lengths and widths for two rectangle objects and draw them by calling the member functions.

```
r1.setlength(3);
r1.setwidth(7);

writeln(' Draw a rectangle: ', r1.getlength(), ' by ' , r1.getwidth());
r1.draw;
new(pr1);
pr1^.setlength(5);
pr1^.setwidth(4);
```

```
writeln(' Draw a rectangle: ', pr1^.getlength(), ' by ' ,pr1^.getwidth());
pr1^.draw;
dispose(pr1);
```

Following is a complete example to show how to use objects in Pascal −

```
program exObjects;
type
   Rectangle = object
   private
      length, width: integer;

   public
      procedure setlength(l: integer);
      function getlength(): integer;

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

      procedure draw;
end;
var
   r1: Rectangle;
   pr1: ^Rectangle;

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.setlength(3);
   r1.setwidth(7);

   writeln('Draw a rectangle:', r1.getlength(), ' by ' , r1.getwidth());
   r1.draw;
   new(pr1);
   pr1^.setlength(5);
   pr1^.setwidth(4);

   writeln('Draw a rectangle:', pr1^.getlength(), ' by ' ,pr1^.getwidth());
   pr1^.draw;
```

```
   dispose(pr1);
end.
```

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

```
Draw a rectangle: 3 by 7
* * * * * * *
* * * * * * *
* * * * * * *
Draw a rectangle: 5 by 4
* * * *
* * * *
* * * *
* * * *
* * * *
```

Visibility of the Object Members

Visibility indicates the accessibility of the object members. Pascal object members have three types of visibility −

Visibility	Accessibility
Public	The members can be used by other units outside the program unit
Private	The members are only accessible in the current unit.
Protected	The members are available only to objects descended from the parent object.

By default, fields and methods of an object are public and are exported outside the current unit.

Constructors and Destructors for Pascal Objects:

Constructors are special type of methods, which are called automatically whenever an object is created. You create a constructor in Pascal just by declaring a method with a keyword constructor. Conventionally, the method name is Init, however, you can provide any valid identifier of your own. You can pass as many arguments as you like into the constructor function.

Destructors are methods that are called during the destruction of the object. The destructor methods destroy any memory allocation created by constructors.

Following example will provide a constructor and a destructor for the Rectangle class which will initialize length and width for the rectangle at the time of object creation and destroy it when it goes out of scope.

```
program exObjects;
type
   Rectangle = object
   private
      length, width: integer;
   public
      constructor init(l, w: integer);
      destructor done;

      procedure setlength(l: integer);
      function getlength(): integer;

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

      procedure draw;
end;

var
```

```
   r1: Rectangle;
   pr1: ^Rectangle;

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

destructor Rectangle.done;
begin
   writeln(' Desctructor Called');
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.init(3, 7);
   writeln('Draw a rectangle:', r1.getlength(), ' by ' , r1.getwidth());
   r1.draw;
   new(pr1, init(5, 4));

   writeln('Draw a rectangle:', pr1^.getlength(), ' by ',pr1^.getwidth());
   pr1^.draw;
   pr1^.init(7, 9);

   writeln('Draw a rectangle:', pr1^.getlength(), ' by ' ,pr1^.getwidth());
   pr1^.draw;
   dispose(pr1);
   r1.done;
end.
```

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

```
Draw a rectangle: 3 by 7
* * * * * * *
* * * * * * *
* * * * * * *
Draw a rectangle: 5 by 4
```

```
* * * *
* * * *
* * * *
* * * *
* * * *
Draw a rectangle: 7 by 9
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
Destructor Called
```

Inheritance for Pascal Objects

Pascal objects can optionally inherit from a parent object. The following program illustrates inheritance in Pascal Objects. Let us create another object named **TableTop**, which is inheriting from the Rectangle object.

```
program exObjects;
type
   Rectangle = object
   private
      length, width: integer;
   public
      procedure setlength(l: integer);
      function getlength(): integer;
      procedure setwidth(w: integer);
      function getwidth(): integer;
      procedure draw;
end;

TableTop = object (Rectangle)
   private
     material: string;
   public
      function getmaterial(): string;
      procedure setmaterial( m: string);
      procedure displaydetails;
      procedure draw;
end;

var
   tt1: TableTop;

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;

function TableTop.getmaterial(): string;
begin
   getmaterial := material;
end;

procedure TableTop.setmaterial( m: string);
begin
   material := m;
end;

procedure TableTop.displaydetails;
begin
   writeln('Table Top: ', self.getlength(), ' by ' , self.getwidth());
   writeln('Material: ', self.getmaterial());
end;

procedure TableTop.draw();
var
   i, j: integer;
begin
   for i:= 1 to length do
   begin
      for j:= 1 to width do
         write(' * ');
   writeln;
   end;
   writeln('Material: ', material);
end;

begin
   tt1.setlength(3);
   tt1.setwidth(7);
   tt1.setmaterial('Wood');
   tt1.displaydetails();
   writeln;
   writeln('Calling the Draw method');
   tt1.draw();
end.
```

Following are the important points which should be noted down −

- The object *Tabletop* has inherited all the members of the Rectangle object.
- There is a draw method in *TableTop* also. When the *draw* method is called using a *TableTop* object, TableTop's draw gets invoked.
- There is an implicit instance named **self** that refers to the current instance of the object.

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

```
Table Top: 3 by 7
Material: Wood

Calling the Draw Method
* * * * * * *
* * * * * * *
* * * * * * *
Material: Wood
```