

Using a Parent Class to create its own children

In the previous article, we used an array of class types to instantiate child classes based on the abstract factory pattern.

Using the same technique, we are going to drop the need for a helper function, such as the **CreateChild** procedure we used in the previous article and we're going to move it into the constructor of our ParentClass.

Consider a class structure something like this

Type

```
ParentClass = class
End;

Child1 = class(ParentClass)
End;

Child2 = class(ParentClass)
End;

Child3 = class(ParentClass)
End;
```

Using the CreateChild helper function, we would utilise something like this:

```
aChild:=CreateChild(ClassName);
```

Instead what we want to do now is something like this:

```
AChild:=ParentClass.Create(ClassName);
```

In this instance the ParentClass is now responsible for returning the correct child instance based on the ClassName parameter. Firstly we're going to overload the constructor with a class function, and we will hide (*optional*) the original constructor as a protected method.

Type

```
ParentClass = class
protected
  constructor Create; overload;
public
  class function Create(ClassName : String) : ParentClass; overload;
End;
```

Now, returning to the previous article, we'll revisit the CreateChild helper function, but this time we will implement it in the class function Create instead. Firstly, we'll create an array of Class of ParentClass as follows:

Type

```
ClassOfParent = class of ParentClass;
```

Const

```
nChildren = 3;
ChildClassArray : Array[1..nChildren] of ClassOfParent =
  (Child1, Child2, Child3);
```

In our pseudo Constructor, we'll implement the previous **CreateChild** function, iterating through the ChildClassArray and comparing the ClassName to the one we wish to create.

```

class function ParentClass.Create(ClassName: String): ParentClass;
Var
    nLoop : Integer;
Begin
    Result:=Nil;
    For nLoop:=1 to nChildren do
        If ChildClassArray[nLoop].ClassName = ClassName then
            Begin
                Result:=ChildClassArray[nLoop].Create;
                Break;
            End;
    End;

```

The calling code is now as simple as

```

Aclass : ParentClass;
Begin
    Aclass:=ParentClass.Create('Child1');
    If Assigned(Aclass) then etc. . .

```

* * * * *

Utilising the second example in article 1, instead of creating a CreateDocument helper function, we could modify parent TDocument class as follows:

```

Type
TDocument = class
    protected
        constructor Create; overload;
    public
        class function Create(Customer : TCustomer) : TDocument; overload;
        class function Supports(Customer : TCustomer) :
Boolean; virtual; abstract;
    End;

```

. . .

```

class function TDocument.Create(Customer: TCustomer): TDocument;
Var
    nLoop : Integer;
Begin
    Result:=Nil;
    For nLoop:=1 to nDocuments do
        if DocumentsArray[nLoop].Supports(Customer) then
            Begin
                Result:=DocumentsArray[nLoop].Create;
                Break;
            End;
    End;

```