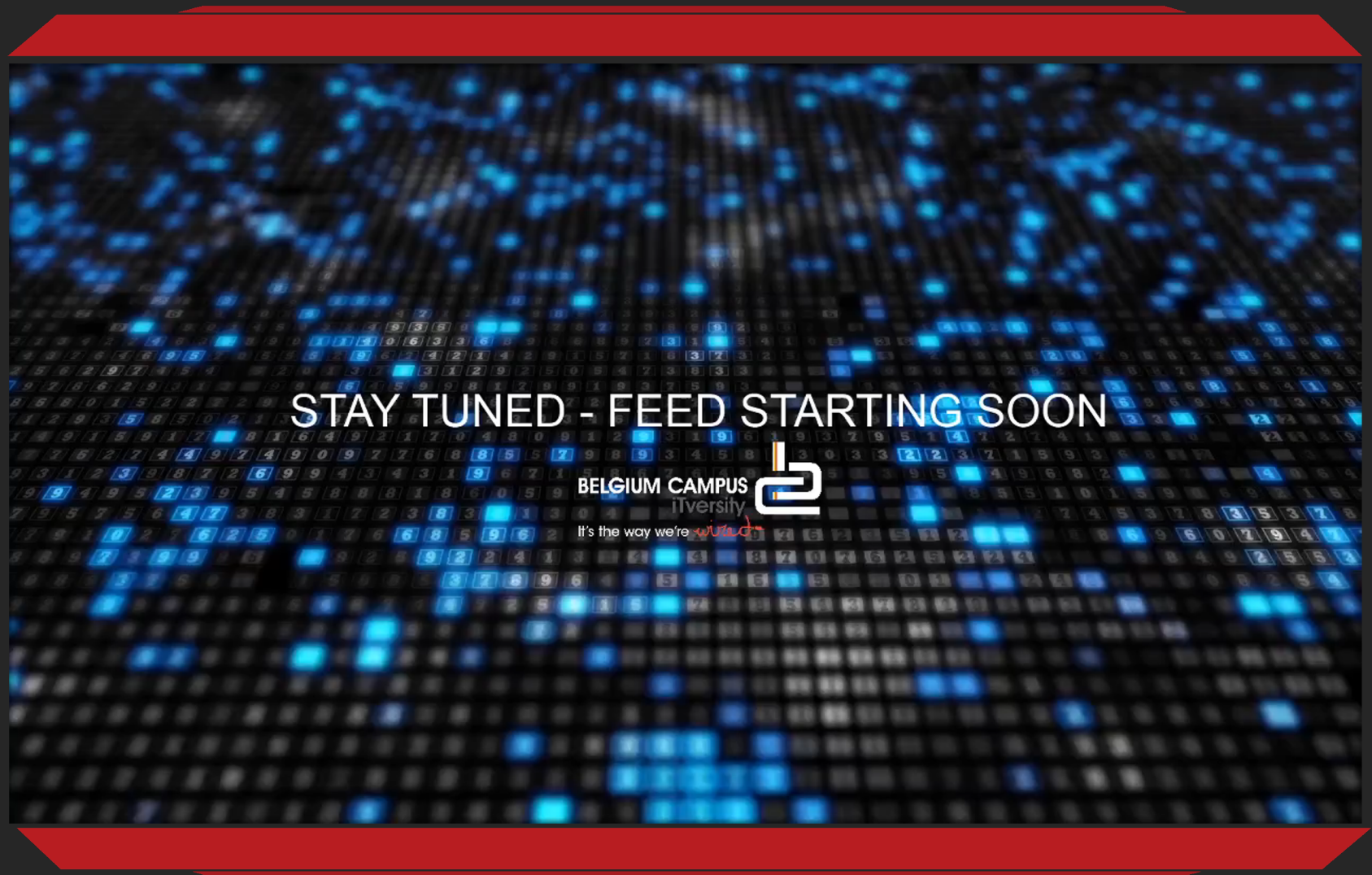

STAY TUNED - FEED STARTING SOON
BELGIUM CAMPUS
iTversity
It's the way we're wired

BELGIUM CAMPUS
iTversity
It's the way we're wired

Delphi - OOP

V. Pretorius

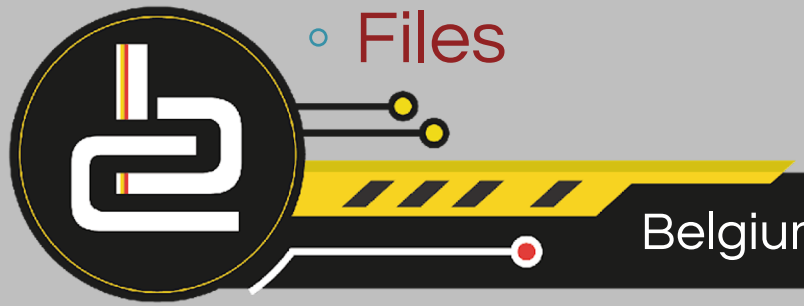
Lesson objectives

- Class declaration
- Object Instantiation
- Encapsulation
- Constructors
- Method Overloading
- Accessors, Mutators

Prior Knowledge

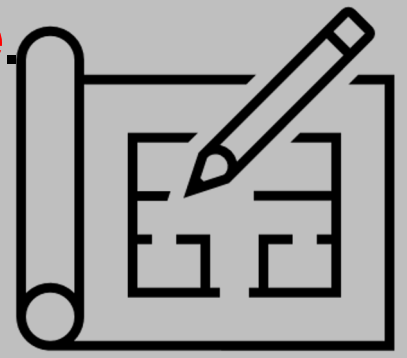
In order to complete the section on **Object Oriented Programming**, prior knowledge about the following is required:

- Scope of variables
- Control structures
- Loops
- Functions
- Procedures
- Arrays
- Files



Classes

The class groups together values that logically belong together in a structure like a new data type.



A class is made up of:

- Fields (private variables that contain data values)
- Methods (procedures and functions)

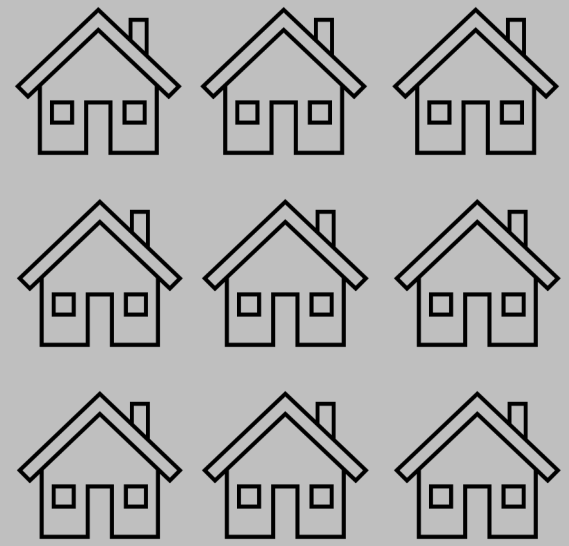
The class is not a variable – you need to declare an object of that type.

Objects

Many objects can be created from the same class, just like many variables can be declared of the same data type.

An object created from a class is called an instance of the class.

One object is a bundle of variables and related functions and procedures.



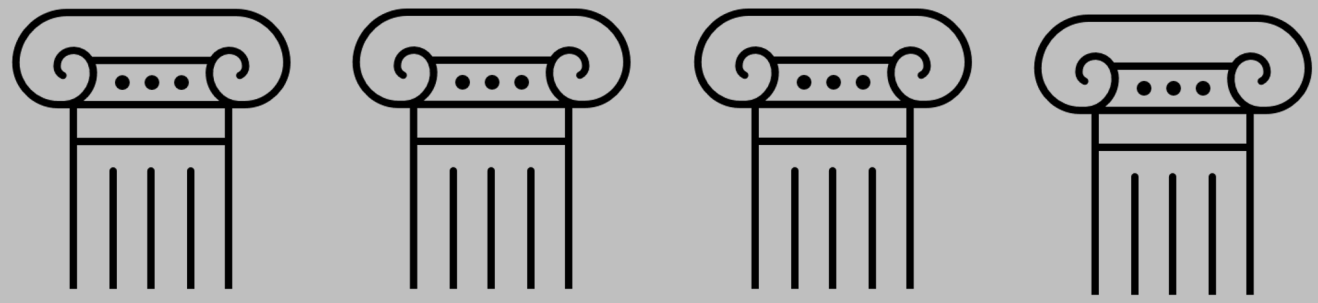
4 Pillars of OOP

Encapsulation

Abstraction

Polymorphism

Inheritance



Encapsulation

enCAPSULate

Private Field

(ingredients)

variables

Mutators methods – procedures to set or change private data in the object.

Accessor methods – functions to **get** or **access** data in the object.

Behavior

(what it does)

Functions & procedures

Object

(one capsule created by using the class)

Class

(the recipe of how to make a capsule)

A class encapsulates the fields and methods. This means that you do not have direct access to these variables. They are PRIVATE. You can change or display the values of fields, only by using the methods of the class.

Default Constructor

Object

```
CerealBox
Private
fL := 0;
fB := 0;
fH := 0;
Public
Function GetAge: integer;
fL := 0;
fB := 0;
fH := 0;
End;
```

Default Constructor: Constructor Create;

You must declare a variable of the Class type first.

Then you have to use a constructor to create an object and set up the fields and methods of an object so that it is ready for use.

You can set all strings to empty fields and numbers to 0 or set the values to any other default values

Application Unit

```
Procedure TForm1.FormCreate(Sender: TObject);
Var CerealBox: TBox;
Begin
  // Instantiate
  CerealBox := TBox.create;
End;
```

Parameterized Constructor

CerealBox

Private

fL := x;

fB := y;

fH := z;

Public

Function GetAge: integer;

fL := a;

fB := b;

fH := c;

End;

Parameterized Constructor: Constructor Create(L, B, H: real);

You can provide your constructor with parameters to initialize fields according to input values.

Application Unit

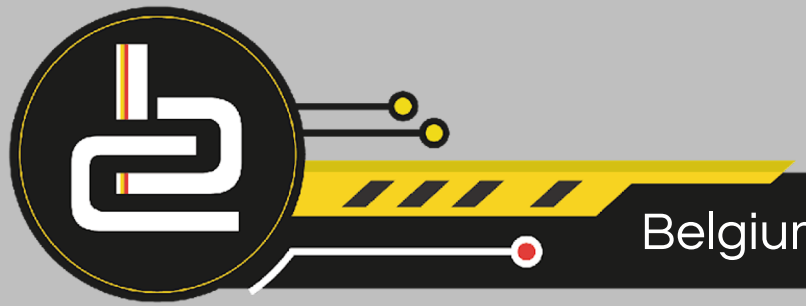
```
Procedure TForm1.FormCreate(Sender: TObject);
Var CerealBox: TBox;
  x, y, z: integer;
Begin
  x := edt1.text; y := edt2.text; z := edt3.text;
  CerealBox := TBox.create(x, y, z);
End;
```

Overloading

When you define two constructors, functions or procedures using the **same name, with different parameters**, you must use overload;

- Constructor Create; overload;
- Constructor Create(x, y, z: integer); overload;

By checking the parameters, the compiler determines which of the routines you are calling.



Accessors

The only way to access the private variables in the object, is to make use of functions or procedures (accessor methods) to access the private variables and make it available in the application.

Class

```
Type
TBox = class(Tobject)
Private
fL,fB,fH : integer; …….
Implementation
Function GetHeight: integer;
Begin
Result:= fH
End;
```

Application Unit


```
Procedure btncall.Onclick)
Var
CerealBox :  Tstudent;
Begin
//instantiate object
CerealBox := TBox.create(5,6,9);

//Function call
lblout.caption
:=inttostr(CerealBox.GetHeight);
```

Mutators

To change the private variables in the object, you make use of procedures (mutator methods) to transfer the new values to the private variables

Class

```
Type TBox = class(Tobject)
Private  fL,fB, fH : integer;
.....
Implementation
Procedure setAll   (a,b,c: integer);
Begin
fL := a; fB := b; fH := c;
End;
Etc.....
```

You can change/calculate the value of private variables after the object was instantiated, using mutator procedures

Application Unit

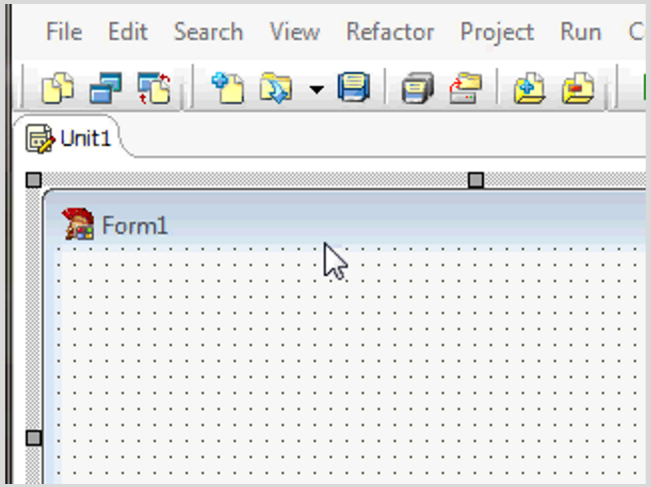
```
Procedure TFrm1.FormCreate(Sender: TObject);
Var  CerealBox :  Tstudent;  x,y,z : integer;
Begin
CerealBox := Tstudent.create;
x := edt16ext; y := edt25ext; z := edt34ext;
                CerealBox.setall ( x, y, z );
```

Example of a question

- Create a class called Tbox with three private variables fH, fL and fB,
  - a parameterised constructor
  - a function getVolume to calculate the volume of the box and
  - a function toString to display the of the box in one line .
- Write a program to
  - create an object and
  - display the Height, Length and Breadth as well as the volume

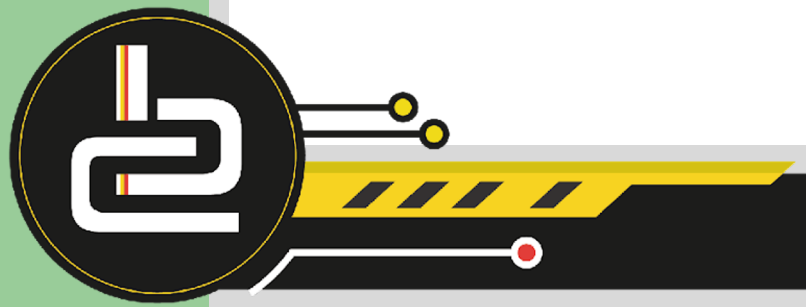
Steps to create class TBox

1. Create a new unit (not an application and not a form).
2. Save the new unit with the prefix cls (in this case clsBox) in the a new folder with an appropriate name, say “Box program”
3. Use “file save as” do not edit the name in the unit



Steps to create a class e.g. TBox

4. Under the word Interface ,
   - Add **uses sysutils**
   - Add the word " **type**"
   - With your cursor after the word "type", use **ctrl j** to generate the skeleton **code for a new class**
   - Use the **prefix f and a**dd attributes under private (fL, fB, fH). (This encapsulates variables and protects the values against unintentional modification)
   - Delete published and protected



Steps to create a class e.g. TBox

5. Under public in the interface (to make methods available in other units that use the class)
   - Add the constructor, function/procedure headings
   - Keep your cursor in the public section and press ctrl-shift-C to put the skeleton code for all subprograms in the implementation section.
   - Note that TBox . is added in front of every function and procedure name

```
type
TBox = class(TObject)
private { private declarations }
  fB,fH,fL : integer;
public  { public declarations }

end;

implementation

{ TBox }
```

Steps to create a class e.g. TBox

6. **Complete the code for the subprograms**
   - The constructor initialises the private variables
   - A function always has a statement result :=

```
constructor TBox.Create(a, b, c:
begin

end;
function TBox.getVolume: integer
begin

end;

end.
```

Steps to create a class TBox

7. The ToString Function is generally used to generate a string representation of all the fields of an object as one string so that they can be displayed easily.

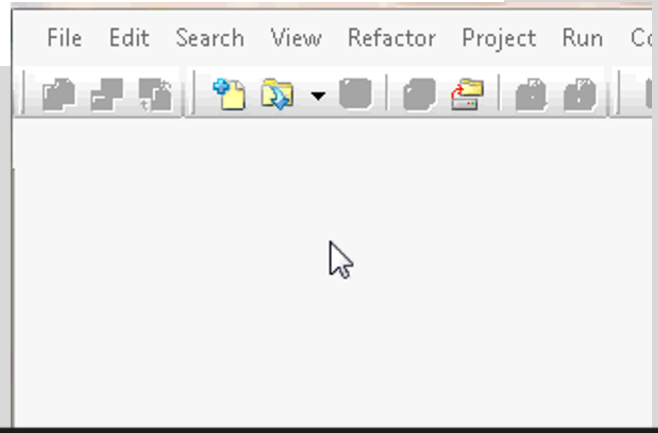
E.g. function TBox.ToString: string;
Begin
Result := inttostr(fB)+#9
+inttostr(fH)+#9
+inttostr(fL)+#9
+inttostr(GetVolume);

```
TBox = class(TObject)
private
  { private declarations }
  fB,fH,fL : integer;
public
  { public declarations }
  constructor Create(a,b,c: integer);
  function getVolume : integer;


published
  { published declarations }
end;
```

Steps to use your class Tbox

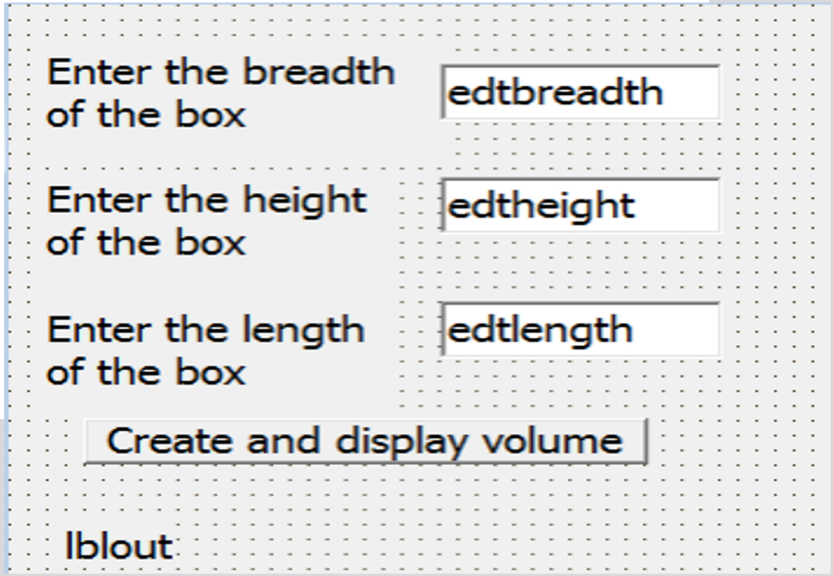
Make a new application

1. save the project and unit in the same folder as the class
2. Add the Name of the unit (clsBox for the Box class), to the Uses section in the application
3. Define a variable MyBox of your class type under private in the application



Steps to use your class TBox

4. Create the user interface - add input components
5. Declare variables and assign input to the variables
6. Create (instantiate) the object using the constructor:
   Mybox := Tbox.create(b,h,l)
   - This is often done in the OnCreate event, but if you use input from edits, you have to do it on a button.



Steps to use your class TBox

7. Call the function(s) of the object to display the values of attributes or calculated values based on the private variables.
8. In the form FormClose event, free the object: mybox.free
   - This will free the memory that was occupie
     running out of memory in large programs

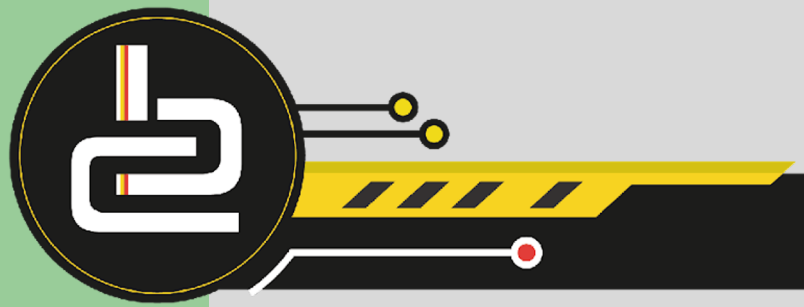
```
procedure TForm1.btnDoClick(Send
var b,h,l : integer;
begin
b := strtoint(edtBreadth.Text);
h := strtoint(edtHeight.Text);
l := strtoint(edtLength.Text);
mybox := TBox.create(b,h,l);

end;
procedure TForm1.FormClose(Sende
begin

end;
```

- Note:
  - All method calls have the format: Objectname.methodname(parameters)
  - Functions are called as part of other statements and procedure calls are statements on their own.

Arrays of objects

- In an array of objects each object can have different values for the fields, but all the objects are of the same type.
- E.g **MyBoxes: array[1..5] of TBox;**

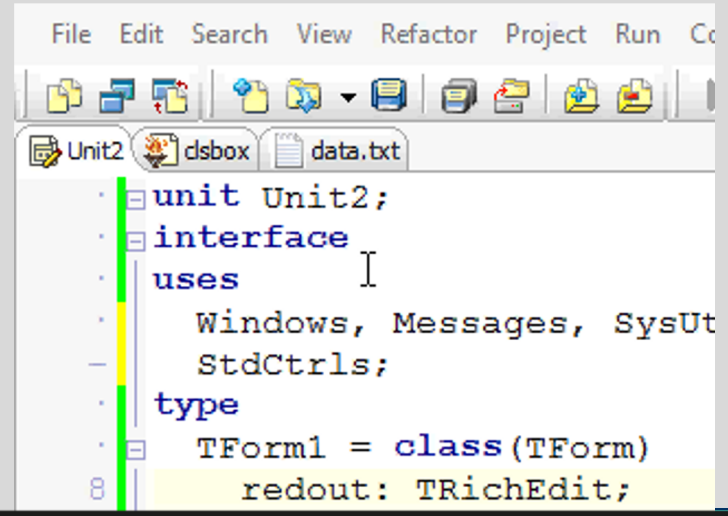
1	fL = 8 fB = 3 fH = 4
2	fL = 7 fB = 5 fH = 6
3	fL = 7 fB = 3 fH = 1
4	fL = 12 fB = 8 fH = 4
5	fL = 9 fB = 3 fH = 4



Steps to re-use your class TBox for an array of Objects

Make a new application

1. Save the project and unit in the same folder as the class
2. Add clsBox to the Uses section in the application
3. Define a variable Myboxes : array[1..5] of Tbox of your class type under private in the application



Steps to re-use your class TBox for an array of Objects

4. Create the user interface - add input components
5. Declare variables
6. Each object in the array must be created:

You can use a for loop

E.g. For i := 1 to 5 do

MyBoxes[i] := TBox.Create;

Or you can create the objects using values from a text file:

```
index := 0;
while not eof(tf) and (index < 5) do begin
   readln(tf,line); ……//separate the values for iBreadth,iHeight and ilength
   inc(index);
   myboxes[index] := TBox.Create(iBreadth,iHeight,ilength);
   end;
```

Animation: creating an array of objects using a text file

Non local variables
myboxes : array[1..5] of TBox; index : integer

```
var tf : textfile;
    line : string;
    ilength, iBreadth,iHeight, i : integer;
    begin
assignfile(tf,'data.txt');
reset(tf);
index := 0;
while not eof(tf) and (index < 5)  do
    begin
    readln(tf,line);
    iBreadth := strtoint(copy(line,1,pos(',',line)-1));
    delete(line,1,pos(',',line));
    iHeight := strtoint(copy(line,1,pos(',',line)-1));
    delete(line,1,pos(',',line));
    ilength :=  strtoint(line);
    inc(index);
    myboxes[index] := TBox.Create(iBreadth,iHeight,ilength);
    end;
closefile(tf);
```

Declare the local variables

Link to the text file

Loop thr... without ... size of a...

Read ... file into ...

Index = 0

Unit2 | dsbox | data.txt

```
8,3,4
7,5,6
7,3,1
12,8,4
9,3,4
```

myBoxes	
1	fB=8 fH= 3 fL=4
2	fB=7 fH= 5 fL=6
3	fB=7 fH= 3 fL=1
4	fB=12 fH= 8 fL=4
5	fB=9 fH= 3 fL=4

Line	8	3,4
iBreadth		
iHeight		
ilength		

Display an array of Objects and free all objects in the array

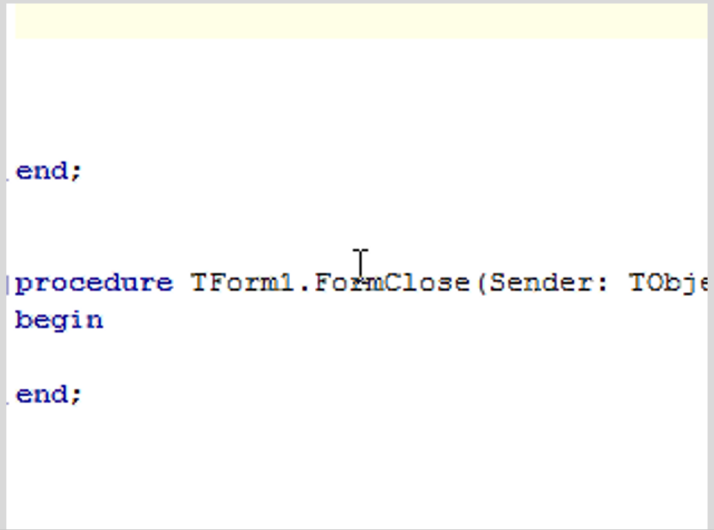
7. Call the functions/procedures of the class to calculate and display the values of variables. You need a **for loop to display** a value for all the objects in the array

e.g

```
redout.Lines.Add('B'+#9+'H'+#9+'L'+#9+'Volume');  //headings
for i := 1 to 5 do
redout.Lines.Add(myboxes[i].tostring)
```

8. In the form OnClose event, free all the objects:

e.g. For i := 1 to 5 do
MyBoxes[i].free



Sorting an array of objects

```
 var j, k : integer;
     Temp : TBox;
begin
for j := 1 to index -1do
     for k := j + 1 to index do
     if myboxes[j]).Getvolume< myboxes[k].GetVolume
 then  begin
          Temp := myboxes[j];
          myboxes[j] := myboxes[k];
          myboxes[k] := Temp;
          end;
```

the temporary variable to swop two values is a single object of the same class

Swop two objects in the array if the condition is true