```
tdaten=record
  name : string;
  alter: integer;
  groesse:real;
end;
```

- Zugriff auf Felder (name, alter, groesse) über externe Methoden, die für jede Unit und jede Variable dieses Records neu geschrieben werden müssen

- Zugriff nicht kontrollierbar (immer Schreib- und Leserechte für jeden)

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public


end;
```

- kein Zugriff auf die Felder mehr möglich von außerhalb der Klasse (**Kapselung**)

- im public-Teil werden nun Methoden zum Lesen und Beschreiben der Felder angelegt, die **geregelten Zugriff** organisieren

ein Beispiel:

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s:string);
end;
```

- mithilfe von set_name() soll es möglich sein, das Feld name zu beschreiben.

- die Implementierung sieht im einfachsten Fall so aus:

```
procedure tdaten.set_name(s:string);
begin
 name:=s;
end;
```

ein Beispiel:

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s:string);
end;
```

- allerdings kann ich das Beschreiben von name jetzt an bestimmte Bedingungen knüpfen, z.B:

```
procedure tdaten.set_name(s:string);
begin
if name<>‘Herbert‘  then
 name:=s;
end;
```

ein Beispiel:

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s,
passwort :string);
end;
```

- man könnte set_name() auch um einen Parameter „passwort“ erweitern, ohne das kein Zugriff möglich ist:

```
procedure tdaten.set_name(s, passwort:string);
begin
if passwort=‘Herbert‘ then
 name:=s;
end;
```

ein Beispiel:

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s:string);
 function get_name : string;
end;
```

- mithilfe von get_name() soll es nun möglich sein, das Feld name zu lesen.
- die Implementierung sieht im einfachsten Fall so aus:

```
function tdaten.get_name :string;
begin
 result:= name;
end;
```

ein Beispiel:

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s:string);
 function get_name : string;
end;
```

- natürlich kann man auch das Lesen von name an Bedingungen knüpfen:

```
function tdaten.get_name :string;
begin
 if name<>‘Herbert‘ then
 result:= name;
end;
```

ein Beispiel:

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s:string);
 function get_name(pw:string)  :
string;
end;
```

- oder wieder nur mit Passwort ein Lesen von name zulassen:

```
function tdaten.get_name (pw:string) :string;
begin
 if pw='Herbert' then
 result:= name;
end;
```

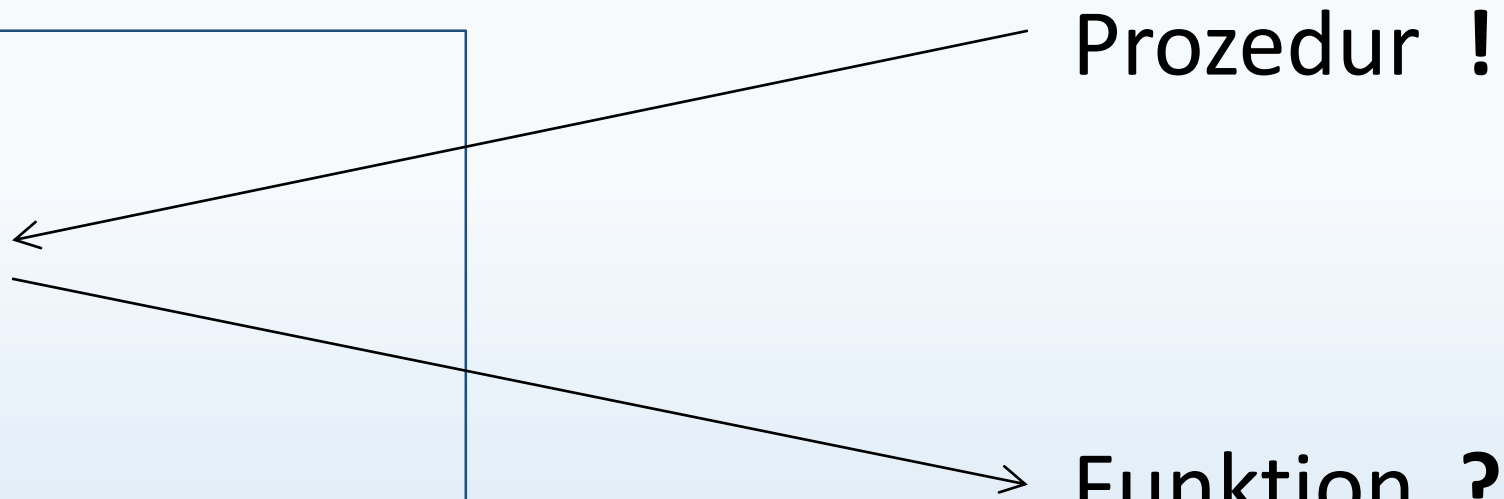
```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s:string);
  function get_name :string;
end;
```

Also merken wir uns:

- Klassenmethoden sind Funktionen und Prozeduren, die den Zugriff auf Felder einer Klasse steuern

- Prozeduren **ÄNDERN** Felder
- Funktionen **LESEN** Felder

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s:string);
  function get_name :string;
end;
```



Praktische Umsetzung:

interface

type

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s:string);
 function get_name : string;
end;
```

der Teil links wird als DEKLARATION bezeichnet und gehört in den Type-Teil des Interface

Praktische Umsetzung:

interface

type

```
tdaten=class
 private
  name : string;
  alter: integer;
  groesse:real;
public
  procedure set_name(s:string);
 function get_name : string;
end;
```

die Erläuterung der Funktions-weise der Methoden wird als IMPLEMENTIERUNG bezeichnet und gehört in den Implementation- Teil der Unit

Praktische Umsetzung:

```
implementation

procedure tdaten.set_name (s:string);
begin
 name:=s;
end;

function tdaten.get_name: string;
begin
 result:=name;
end;
```

beide Methoden gehören zur Klasse tdaten und werden daher durch Voranstellen des Klassennamens der Klasse zugeordnet

Verwenden der Klasse tdaten

- tdaten ist ein **Datentyp** wie integer, real oder string

- d.h. die Klasse kann nicht so verwendet werden: `tdaten.set_name('Peter');`

  genauso wenig wie ich schreiben kann: `integer:=5;`

→ Ich muss eine Variable des Typs tdaten erzeugen:

```
var
    person1: tdaten
```

Verwenden der Klasse tdaten

```
var
    person1: tdaten
```

- dennoch kann ich jetzt mit person1 noch nicht arbeiten!
- Variablen von Klassen (= Objekte) müssen explizit erzeugt werden
- dies funktioniert über die Klassenmethode **create**

person1 := Tdaten.create;

- jede Klasse besitzt die Methode **create** automatisch
- man nennt sie **Konstruktor** der Klasse
- später werden wir sehen, dass man diese Methode auch selbst schreiben und verändern kann

Verwenden der Klasse tdaten

- person 1 kann ich nun nutzen: **person1.set_name(‘Peter‘);**

oder: **s:= person1.get_name;**
oder
showmessage (person1.get_name);

- nicht funktionieren wird: **person1.name:=‘Peter‘;**

Aber wieso nicht ???

Verwenden der Klasse tdaten

var

person1: tdaten

Variablen von Klassen werden als **OBJEKTE** bezeichnet.
Objekte sind **INSTANZEN** einer Klasse, d.h. sie werden gemäß der Deklaration der Klasse mit dem Konstruktor erzeugt nach dem Muster

Objekt := Klasse.create;

So hat **person1** die Felder name, alter und groesse sowie
die Methoden get_name und set_name() von seiner Klasse Tdaten übernommen. Wie in der Deklaration sind die Felder weder direkt lesbar oder beschreibbar außerhalb der Klasse (**private**).

Arbeitsauftrag:

1) Lege eine neues Projekt mit zwei Units an : Ein GUI und eine unit klassen.
2) Binde die unit klassen im uses-Teil des GUI ein.
3) Deklariere wie oben erklärt die Klasse TAuto mit den Feldern
   marke, typ (=string) , baujahr, kmstand(=integer) und preis (=real).
4) Deklariere und implementiere für alle Felder die Methoden für den Zugriff
   (get – und set –Methoden, alle unter **public**)
5) Lege in der GUI das Objekt Porsche911 als Variable an und erzeuge das Objekt mit
   dem Konstruktor.
6) Weise dem Objekt Porsche911 folgende Eigenschaften zu:

   marke: Porsche
   typ:911
   baujahr: 2012
   kmstand: 78354
   preis: 58634,99