EIGHTEEN

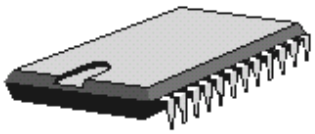
An Introduction to Object Oriented Programming

Object oriented programming is a new style of writing programs which has a number of important advantages, particularly when developing large and complex software projects.

You have already been using ***objects*** in every Delphi program you have written, although you may not have realised this. Each **component** is an ***object*** - for example: *buttons*, *spin edit boxes*, *string grids*, *image boxes*, and *forms* themselves.

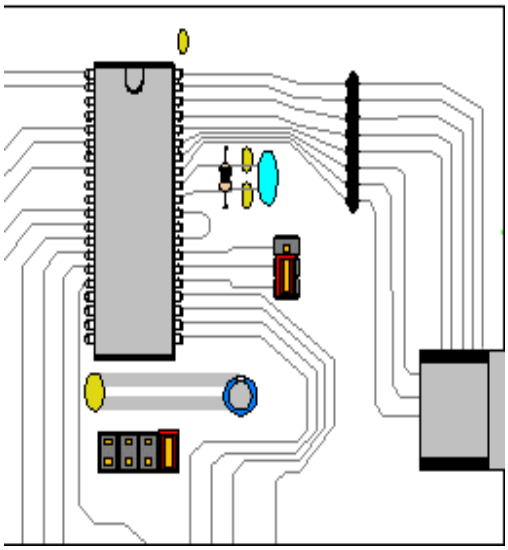
An easy way to understand object oriented programming is to make an analogy with electronics:

Electronic circuits are made up of components:

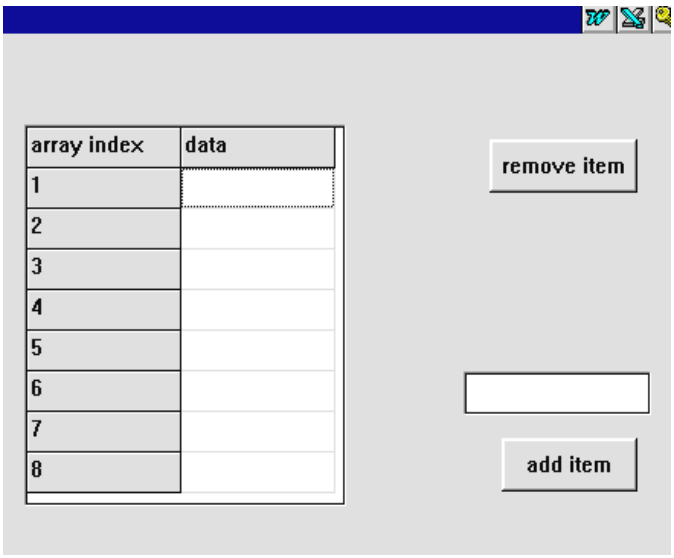


Programs are made up of components:

Electronic components are assembled together to produce complete devices:

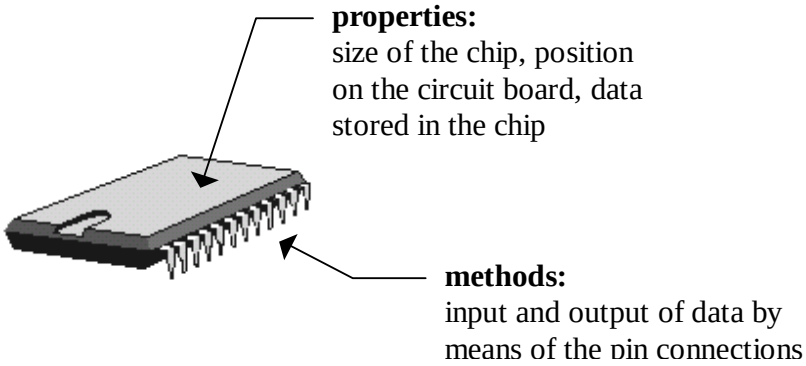


Program components are assembled together to produce software applications:



Think now about the purpose of an

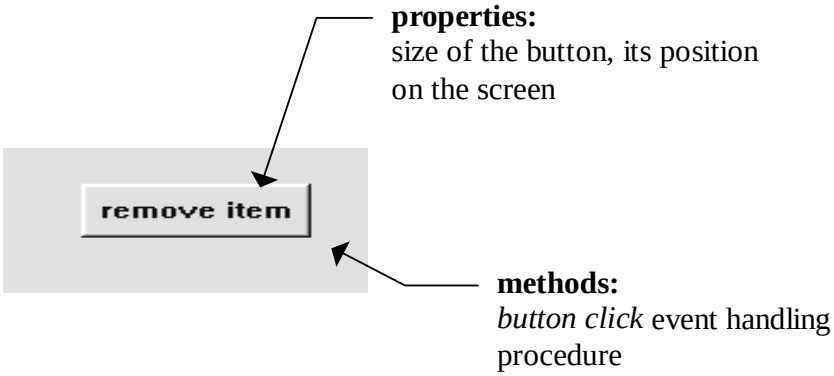
electronic component - for example a memory chip for a computer:



To understand how the chip forms part of the overall computer system, we need to know:

- its ***properties*** - a description of its physical size and position on the circuit board, the memory capacity, and the data currently stored in it.
- the ***methods*** by which it carries out actions or communicates with the rest of the computer - in this case, the input and output of data via the metal pins.

In ***Object Oriented Programming***, the objects making up the program also have properties and methods:



Every object has:

- **properties**, which describe the object.
  {For a *Button*, these include: width, height, the caption displayed, the position on the Form, and whether or not it is visible.}

- **methods**, which carry out actions or communicate with the rest of the program.
  {For example: an event handling procedure to carry out some data processing each time the button is clicked.}

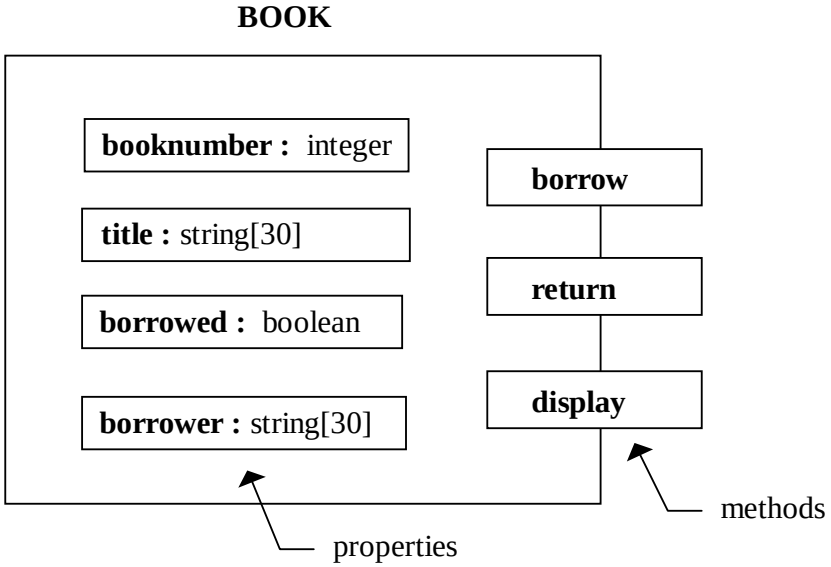
In addition to the component objects already available, ***Delphi also allows us to add our own objects to a program***. In the next section we will see how this is done in a simple example program:

A computer studies lecturer has a collection of 8 reference manuals which can be borrowed by students. The lecturer requires a computer program to record the names of the borrowers when the manuals are out on loan.

Begin the program by setting up a directory BOOKS and save a Delphi project into it. Use the Object Inspector to **maximize** the *Form*, and drag the Form grid to nearly fill the screen.

The objects used in a program can be any kind of building blocks which help towards a solution of the programming problem, so in this case we might choose to set up '**book**' objects.

We begin by considering what **properties** and **methods** the book objects should have. To help with this, an ***object structure diagram*** can be drawn:



The ***properties*** are shown inside the object structure diagram, and the ***methods*** are shown as rectangles cutting through the edge of the diagram - this reflects the idea that *properties* belong personally to an object, but *methods* provide ways for the object to communicate with the outside program.

In the context of our simple library program, the important *properties* of a book are:
- The **book number** and **title**, which allow us to identify a particular book.
- Whether or not the book has been **borrowed.**
- If the book *is* out on loan, the name of the **borrower**.

To operate the program we will need *methods* which can:
- **Record a book being borrowed,** by setting *borrowed* to true and entering the name of the borrower.

- **Record a book being returned,** by setting *borrowed* to false**.**
- **Display a list of the books,** showing which are out on loan and the names of the borrowers .

The first step in creating an object is to set up a **class definition**. {If we again use the electronics analogy, this is equivalent to producing a design for the way the electronic component will be manufactured.}

Go to the '**type**' heading near the top of *Unit1* and add the class definition for a book :

```
type
  TBook = class(TObject)
    booknumber:integer;
    title:string[30];
    borrowed:boolean;
    borrower:string[30];
    procedure borrow;
    procedure return;
    procedure display;
  end;

  TForm1 = class(TForm)
       ........
```

Notice that object class definitions usually begin with the letter '***T***' and list all the properties and methods which will be needed.

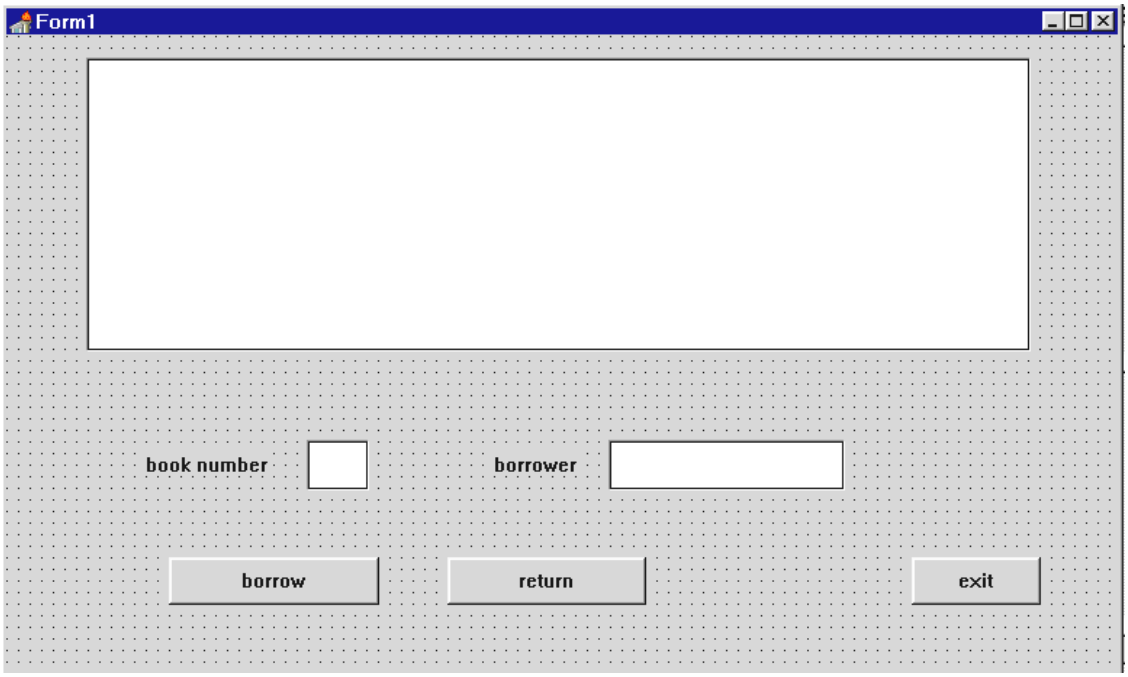
Now move down to the implementation section of the program and add empty procedures for the ***borrow***, ***return***, and ***display*** methods - we will come back later to add lines of program to these procedures:

```
implementation
{$R *.DFM}

  procedure TBook.borrow;
  begin
  end;

  procedure TBook.return;
  begin
  end;

  procedure TBook.display;
  begin
  end;

end.
```

So far, we have not actually produced any objects. There are 8 computer manuals in the library, so an array of 8 ***book*** objects will be needed. Go to the '***variables***' heading just above the *implementation* section and add this array:

```
var
  Form1: TForm1;
  book:array[1..8] of TBook;

implementation
   ......
```

Go now to the ***Form1*** window and add components to the grid as shown:



- Place a ***list box*** in the upper part of the screen.
- Below this add two ***edit boxes***, and place ***labels*** alongside with the captions '**book number**' and '**borrower**'.
- At the bottom of the screen put three ***buttons*** with the captions '**borrow**', '**return**' and '**exit**'.

Double-click the '**exit**' button to produce an event handler, then add a *halt* command:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  halt;
end;
```

Compile and run the program to check that the components are displayed correctly, then click the '**exit**' button to return to the Delphi editing screen.

We are now ready to initialise the program by setting up the book titles. Double-click the mouse on the *Form1* grid to produce an '**OnCreate**' event handler. Add the lines:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i:integer;
begin
  for i:=1 to 8 do
  begin
    book[i]:=TBook.create;
    book[i].booknumber:=i;
    book[i].borrowed:=false;
  end;
  book[1].title:='HTML for Web pages';
  book[2].title:='Windows 95 reference';
  book[3].title:='Microsoft Office 97';
  book[4].title:='3-D graphics techniques';
  book[5].title:='C++ object oriented programs';
  book[6].title:='Using Java';
  book[7].title:='Databases in Delphi';
  book[8].title:='80486 machine code';
  listbox1.clear;
  for i:=1 to 8 do
    book[i].display;
end;
```

This procedure uses a loop to create the 8 book objects in the array:

```
for i:=1 to 8 do
begin
   book[i]:=TBook.create;
```

We also make use of the loop counter to initialise the book numbers:

```
book[i].booknumber:=i;
```

and the ***borrowed*** property for each book is set to *false*:

```
book[i].borrowed:=false;
```

We then enter the book titles:

```
book[1].title:='HTML for Web pages';
book[2].title:='Windows 95 reference';
      . . . . . .
```

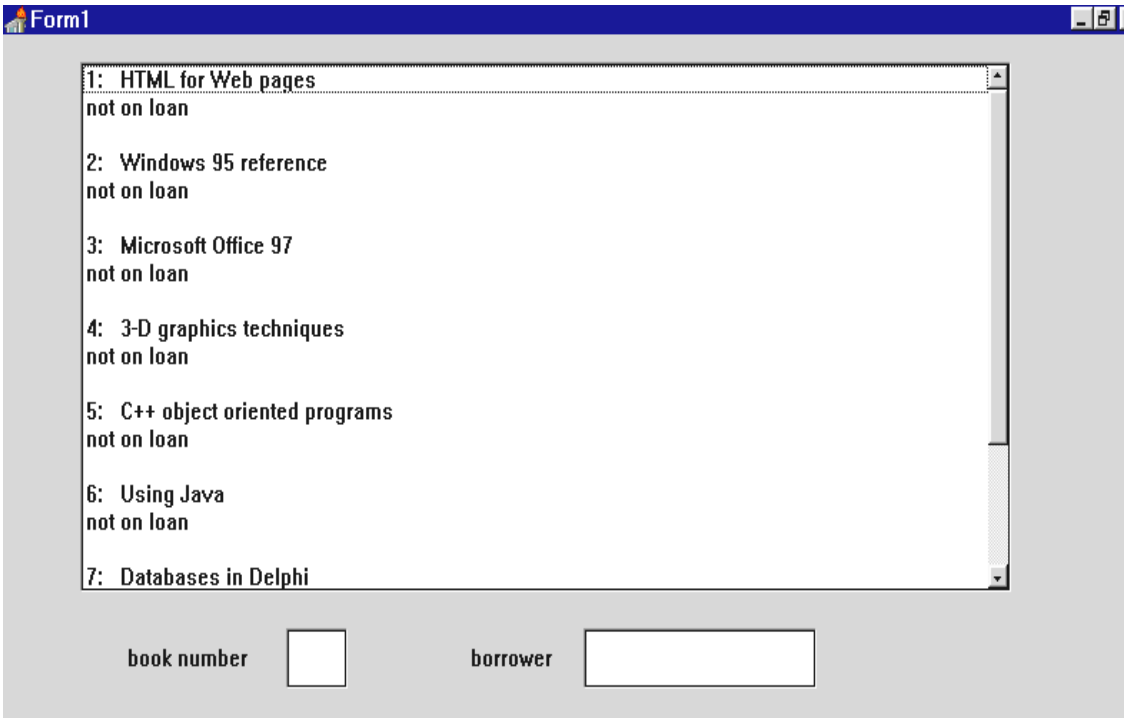
Finally, the ***display*** method is used to show this book information in the list box:

```
listbox1.clear;
for i:=1 to 8 do
     book[i].display;
```

Go now to the ***display*** method, and add the lines of program:

```
procedure TBook.display;
var
  textline:string;
begin
  textline:=inttostr(booknumber);
  textline:=textline+ ':   ' + title;
  Form1.listbox1.items.add(textline);
  if borrowed=false then
    Form1.listbox1.items.add('not on loan')
  else
  begin
    textline:='on loan to ' + borrower;
    Form1.listbox1.items.add(textline);
  end;
  Form1.listbox1.items.add('');
end;
```

Compile and run the program. Check that the ***book numbers*** and ***titles*** are displayed correctly, and that each book is shown as ***not on loan***:



Return to the Delphi editing screen. Double-click the '**borrow**' button to set up an event handler, then add the lines:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  n,i:integer;
begin
  n:=strtoint(edit1.text);
  if (n>=1) and (n<=8) and (edit2.text>'')then
  begin
    book[n].borrow;
    listbox1.clear;
    for i:=1 to 8 do
      book[i].display;
  end;
  edit1.text:='';
  edit2.text:='';
end;
```

This procedure reads the book number which has been entered in *edit box 1* and converts it to an integer number:

n:=strtoint(edit1.text);

We check that this is a valid book number in the range 1 - 8, and also that the name of a borrower has been entered in *edit box 2*:

if (n>=1) and (n<=8) and (edit2.text >' ') then . . . .

If these checks are successful, we call the ***borrow*** method for the appropriate book and record the loan:

book[n].borrow;

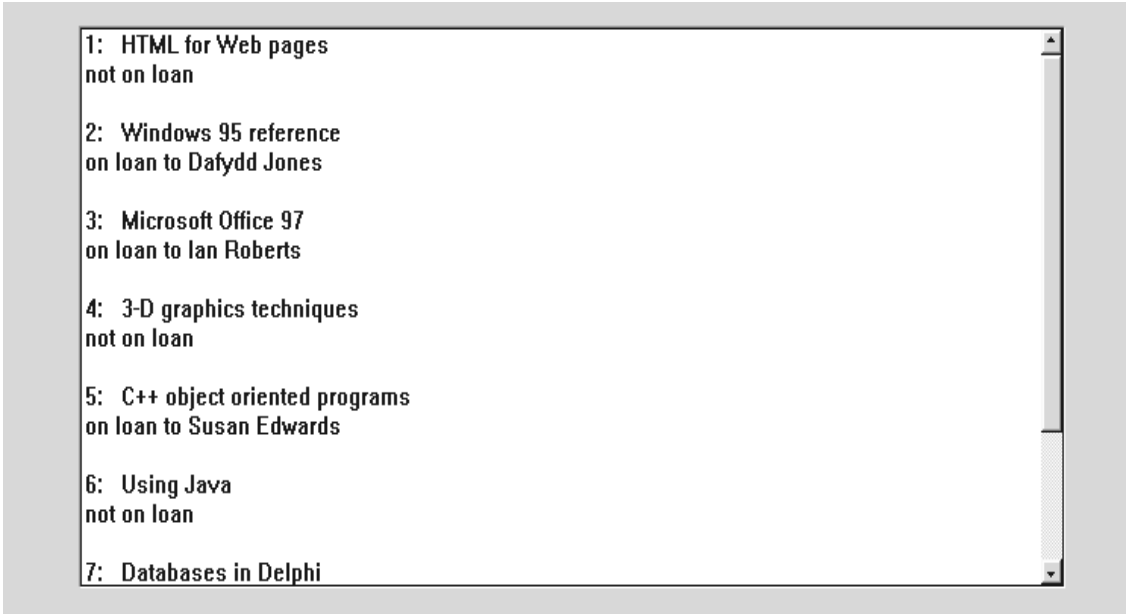
The information in the list box is then redisplayed to show the loan:

```
listbox1.clear;
for i:=1 to 8 do
  book[i].display;
```

Go to the ***borrow*** procedure and add the lines of program to record the loan:

```
procedure TBook.borrow;
begin
  borrowed:=true;
  borrower:=Form1.edit2.text;
end;
```

Compile and run the program. Check that loans can be recorded correctly, as shown below, then return to the Delphi editing screen.



To complete the program, it is necessary to handle the return of books. Double-click the '**return**' button and add the lines of program:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  n,i:integer;
begin
  n:=strtoint(edit1.text);
  if (n>=1) and (n<=8) then
  begin
    book[n].return;
    listbox1.clear;
    for i:=1 to 8 do
      book[i].display;
  end;
  edit1.text:='';
end;
```

Add lines to the ***return*** procedure to update the record:

```
procedure TBook.return;
begin
  borrowed:=false;
  borrower:='';
end;
```

Compile and run the finished program. Check that data is displayed correctly when books are borrowed or returned.

For our second program using object oriented methods, we will look at a more complicated problem where more than one kind of object is required:

Theatre bookings

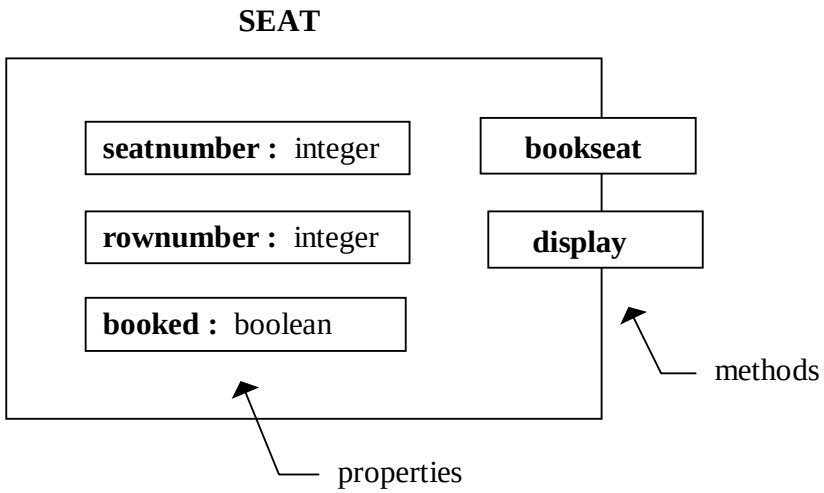
A theatre requires a computer program to record the seat bookings for a play. The theatre has 8 rows each containing 12 seats. Initially, all the seats are shown as unbooked.

It should be possible to enter the number of the row and the quantity of seats required, and the computer will make the booking if sufficient seats are available.

(To keep the program as simple as possible, we will not worry about the cancellation of bookings.)

Begin the program by setting up a directory THEATRE and save a Delphi project into it. Use the Object Inspector to **maximize** the *Form*, and drag the Form grid to nearly fill the screen.

Remember that objects can be any kind of building blocks which help towards solving the problem, so in this case we might choose to set up a '**seat**' object. Before starting the programming, we draw an ***object structure diagram*** for a seat:



The important *properties* for a seat are:

- The **seat number**, which allows us to identify a particular seat in the row.
- The **row number** in which the seat is situated
- Whether or not the seat is **booked.**

To operate the booking system, we will need *methods* which can:

- **Book** the **seat**.
- **Display** the seat booking information on screen.

We need to write a class definition for the object. Do this below the '**type**' heading near the top of *Unit 1*:

```
type
  TSeat = class(TObject)
    seatnumber:integer;
    rownumber:integer;
    booked:boolean;
    procedure bookseat;
    procedure display;
  end;

  TForm1 = class(TForm)
         ........
```

Add empty procedures below the ***implementation*** heading. We will return later to complete these:

```
implementation
{$R *.DFM}

procedure TSeat.bookseat;
begin
end;

procedure TSeat.display;
begin
end;
```

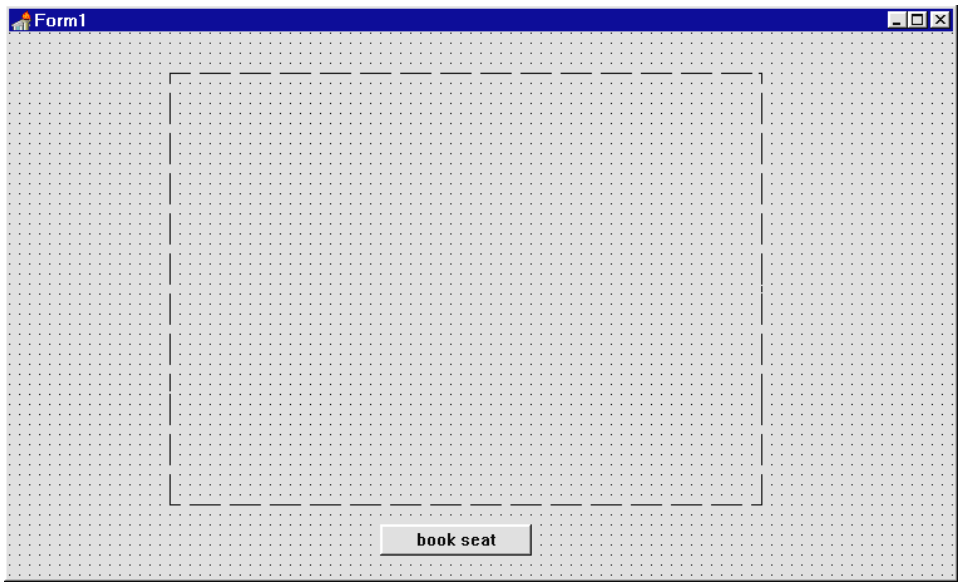
Notice that each procedure heading includes '**TSeat**' to show that it belongs to the '**seat**' object.

We will begin by setting up and testing a single seat object - once this is working correctly, we can add further seats to make up the complete theatre. Allocate a variable name under the ***var*** heading:

```
var
  Form1: TForm1;
  seat: TSeat;
```

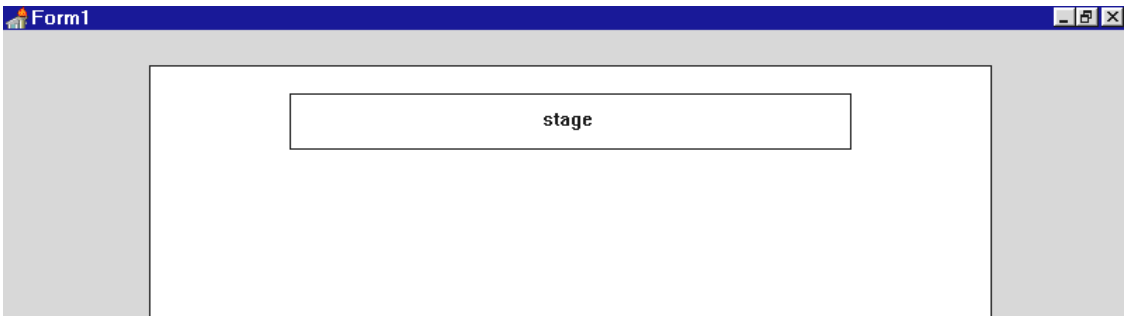
We can now turn our attention to the screen display. As a neat way of showing the bookings, we could draw a plan of the theatre. Booked seats can be coloured red and unbooked seats coloured green. Bring the Form1 window to the front, then add an ***Image box*** as shown below. Set the **height** property to **400** and the **width** property to **600**.

Put a ***Button*** below the image box and give this the caption '**book seat**':



Double-click the Form grid outside the image box to produce an OnCreate procedure. Add lines of program which will draw a white background for the theatre plan and add a rectangle to show the position of the stage:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with image1.canvas do
  begin
    brush.color:=clWhite;
    rectangle(0,0,600,400);
    rectangle(100,20,500,60);
    textout(280,30,'stage');
  end;
end;
```



Compile and run the program to test the display, then return to the Delphi editing screen.

We can now initialise and display the seat. Add lines to the ***OnCreate*** event handler:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with image1.canvas do
  begin
    brush.color:=clWhite;
    rectangle(0,0,600,400);
    rectangle(100,20,500,60);
    textout(280,30,'stage');
  end;
  seat:=TSeat.create;
  seat.seatnumber:=1;
  seat.rownumber:=1;
  seat.booked:=false;
  seat.display;
end;
```

Return to the TSeat.display procedure and add the lines:

```
procedure TSeat.display;
var
  xpos,ypos:integer;
begin
  with Form1.image1.canvas do
  begin
    if booked then
      brush.color:=clRed
    else
      brush.color:=clLime;
    xpos:=100+seatnumber*30;
    ypos:=70+rownumber*30;
    rectangle(xpos,ypos,xpos+20,ypos+20);
  end;
end;
```

The first section of the procedure sets the colour code for the seat - red if booked and green if unbooked:

```
if booked then
 brush.color:=clRed
else
 brush.color:=clLime;
```

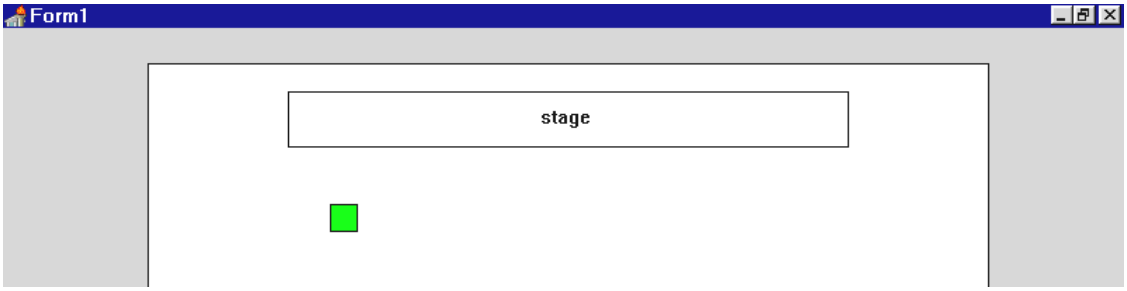
We then use the seat number and row to calculate a suitable position to display this seat on the plan of the theatre:

xpos:= 100 + seatnumber*30;
ypos:= 70 + rownumber*30;

Finally, we plot a rectangle to represent the seat:

rectangle(xpos,ypos,xpos+20,ypos+20);

Compile and run the program. A green rectangle should appear, indicating that initially the seat is not booked:



Return to the Delphi editing screen. Double-click the '**book seat**' button to create an event handler. Add lines to call the *method* which books the seat, then the *method* which redisplays it on the theatre plan:

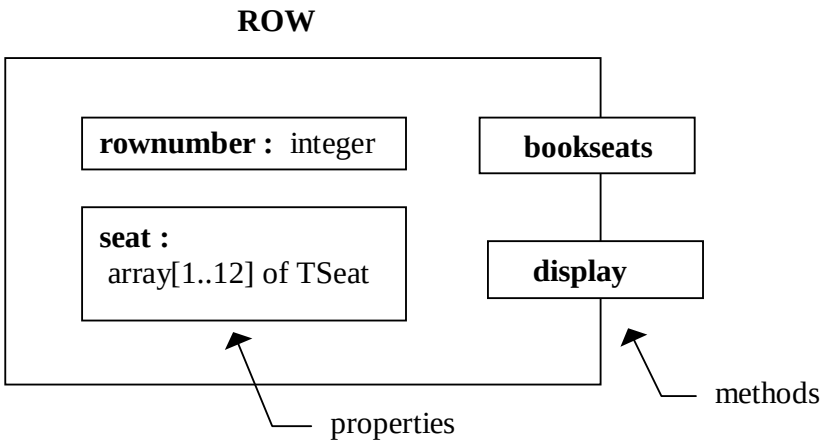
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  seat.bookseat;
  seat.display;
end;
```

Add a line to the **TSeat.bookseat** procedure to change the ***booked*** variable to true:

```
procedure TSeat.bookseat;
begin
  booked:=true;
end;
```

Compile and run the program again. This time click the '**book seat**' button and the rectangle representing the seat should change to red. (In this example program we will not worry about cancelling bookings, but an option to do this would be needed in a complete booking system.)

We have produced and tested a single seat object. A convenient way to move forward now is to define an object called '**row**' which is made up of 12 seats. An *object structure diagram* for **row** is given below:



The *properties* of a **row** are:

- Its **number**, which tells us how far back it is situated in the theatre.
- The group of **12 seats** which make up the row, some of which may be booked and others unbooked.

The *methods* we will require are:

- A procedure to book a group of seats in the row (if sufficient are available).
- A procedure to display the row of seats on the theatre plan, colour coded to show whether or not they are booked.

Add a class definition for the ***row*** object below the **TSeat** definition**:**

```
TRow = class(TObject)
   rownumber:integer;
   seat:array[1..12] of TSeat;
   procedure bookseats;
   procedure display;
 end;

 TForm1 = class(TForm)
     ..........
```

Go to the **var** heading. Delete the entry for **seat** and replace this with **row**. Seats are now part of the **row** object and do not need to be shown as separate variables:

```
var
  Form1: TForm1;
  row: TRow;
```

Modify the **FormCreate** procedure as shown below:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  n:integer;
```

```
begin
  with image1.canvas do
  begin
    brush.color:=clWhite;
    rectangle(0,0,600,400);
    rectangle(100,20,500,60);
    textout(280,30,'stage');
  end;
  row:=TRow.create;
  row.rownumber:=1;
  for n:=1 to 12 do
  begin
    row.seat[n]:=TSeat.create;
    row.seat[n].seatnumber:=n;
    row.seat[n].rownumber:=1;
    row.seat[n].booked:=false;
  end;
  row.display;
end;
```

The purpose of these changes are to initialise *all* the seats in row 1. The loop allocates the seat numbers, and sets each seat to be unbooked. We finally call the **display** method to plot the seats on the theatre plan.

Set up a blank procedure for the **TRow.bookseats** method:

```
procedure TRow.bookseats;
begin
end;
```

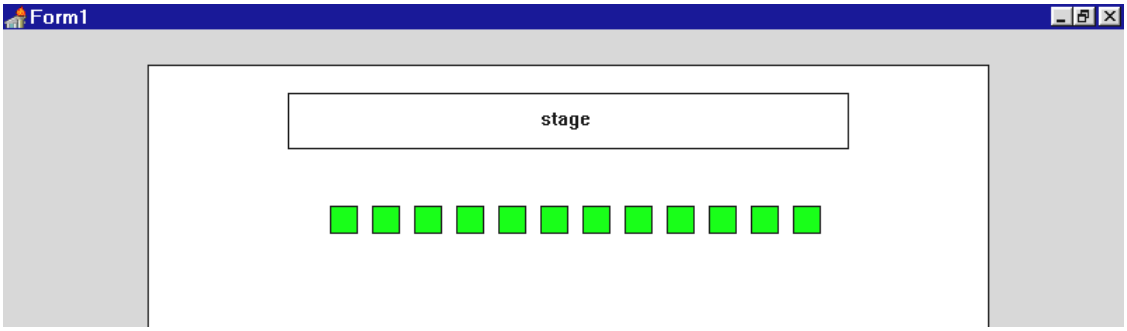
Delete the lines from the '**book seat**' button click procedure, leaving just the heading and the begin...end pair. We will come back to complete this procedure later:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
```

We can now add the **TRow.display** method to plot the seats on the theatre plan. This makes use of the procedure we wrote earlier to display a single seat, but calls this 12 times:

```
procedure TRow.display;
var
  i:integer;
begin
  for i:=1 to 12 do
    seat[i].display;
end;
```

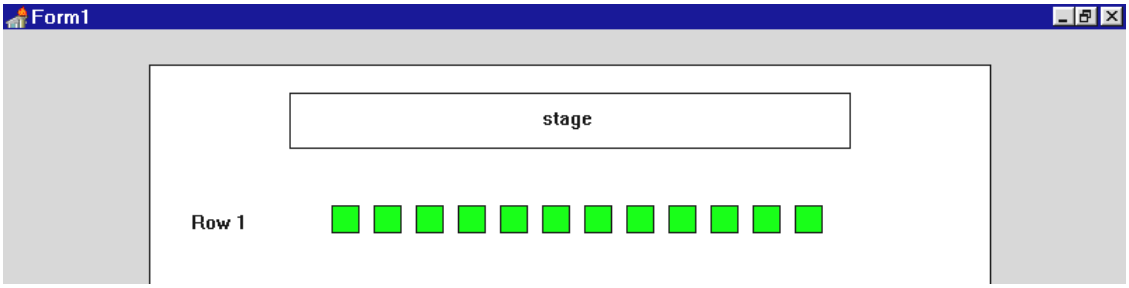
Compile and run the program. A complete row of 12 seats should now be displayed. All will be coloured green as they are initially unbooked:



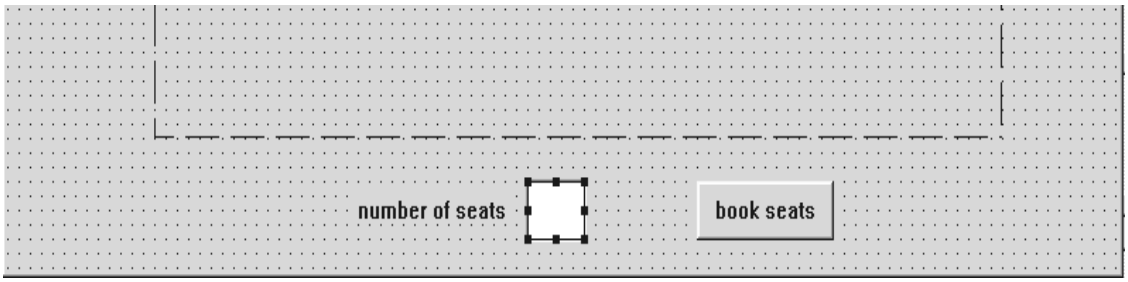
It would be useful to also display the row number on the plan. To do this, return to the Delphi editing screen and add lines to the **TRow.display** procedure:

```
procedure TRow.display;
var
  i,ypos:integer;
  textline:string;
begin
  textline:='Row '+inttostr(rownumber);
  ypos:=74+rownumber*30;
  with Form1.image1.canvas do
  begin
    brush.color:=clWhite;
    textout(30,ypos,textline);
  end;
  for i:=1 to 12 do
    seat[i].display;
end;
```

Compile and run the program. The row number should now appear:



Return to the Delphi editing screen. We must now devise a way to enter bookings. Add an ***edit box*** to the Form, and a ***label*** alongside with the caption '**number of seats**'. Change the caption on the *button* to '**book seats**':



Double-click the edit box to produce an event handler. Add lines of program to record the number of seats wanted:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  if edit1.text='' then
    wanted:=0
  else
    wanted:=strtoint(edit1.text);
end;
```

Add the variable '**wanted**' to the *Public declarations* section:

```
public
    { Public declarations }
    wanted:integer;
```

Compile and run the program. Check that the edit box is error trapped to accept only integer numbers, then return to the Delphi editing screen.

Double-click the 'book seats' button to show the event handler, then add the program lines:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  count,i:integer;
begin
  count:=0;
  for i:=1 to 12 do
  begin
    if row.seat[i].booked=false then
      count:=count+1;
  end;
  if count>=wanted then
  begin
    count:=0;
    i:=0;
    repeat
      i:=i+1;
```

```
        if row.seat[i].booked=false then
        begin
          row.seat[i].booked:=true;
          count:=count+1;
        end;
      until count=wanted;
      row.display;
    end;
  end;
```

This procedure can be explained using an algorithm progressive refinement sequence. The basic tasks to be carried out are:

1. check if sufficient seats are available
2. IF sufficient seats THEN
3. make the booking
4. redisplay the theatre plan
5. END IF

Step 1 can be further refined:

1.1 set *count* to zero
1.2 LOOP for each of the 12 seats
1.3 IF the current seat is not booked THEN
1.4 add 1 to *count*
1.5 END IF
1.6 END LOOP

At the end of step 1, *count* will tell us how many seats in the row are not booked. This can then be compared to the number of seats wanted by the customer. If there are sufficient seats available, we move on to step 3 to make the booking.

Step 3 can be shown in more detail:

3.1 set *count* to zero - {this will record the number of seats we book}
3.2 set a pointer to zero - {this pointer will move along the row as each seat is being checked}
3.3 LOOP
3.4 move the pointer forward by 1
3.5 IF the seat at the current pointer position is not booked THEN
3.6 book the seat
3.7 add 1 to *count*
3.8 END IF
3.9 UNTIL *count* is equal to the number of seats required

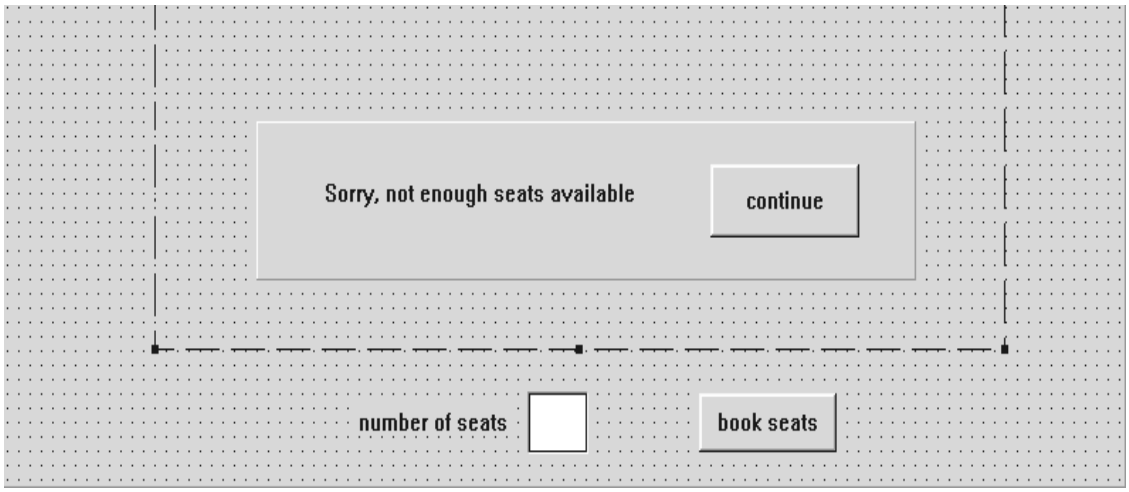
Compile and run the program. Use a test sequence to check that the seat booking procedure works correctly. For example, make the following sequence of entries:

1 seat wanted booking accepted
3 seats wanted booking accepted

5 seats wanted	booking accepted
4 seats wanted	booking not possible
2 seats wanted	booking accepted

When you are convinced that the program works correctly, return to the Delphi editing screen.

At present the computer just ignores booking requests if there are not enough seats available. It would be better to display a message for the user. Add a ***panel*** to the form. Use the *object inspector* for the panel to set the **visible** property to **false**, and blank out the caption.



Add a ***label*** with the caption '**Sorry, not enough seats available**', and a ***button*** with the caption '**continue**'.

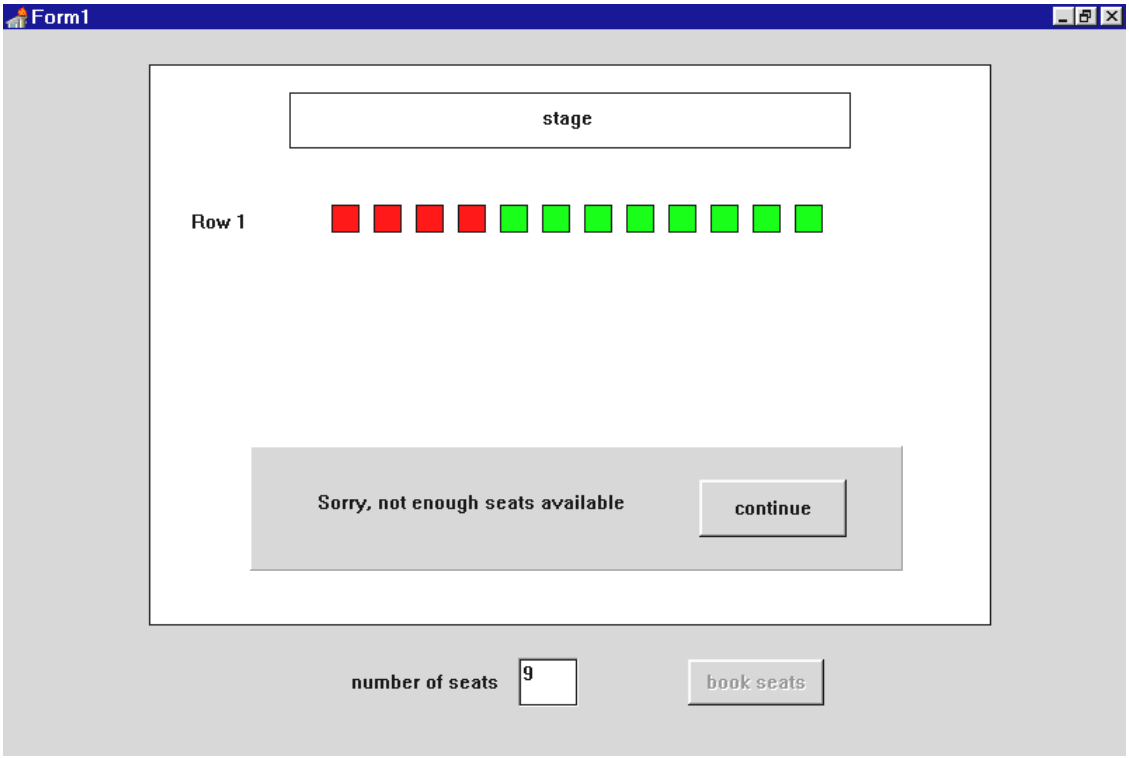
Double-click the '**continue**' button to create an event handler and add the lines:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  panel1.visible:=false;
  button1.enabled:=true;
end;
```

Return to the '**book seats**' buutton click procedure and add an '**else**' condition to display the panel if a booking cannot be made:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  count,i:integer;
begin
   ........
   if count>=wanted then
   begin
      .......
    row.display;
  end
  else
  begin
    button1.enabled:=false;
    panel1.visible:=true;
  end;
end;
```

Compile and run the program. Make a booking for 4 seats, then try to book a further 9 seats. The panel and warning message should appear:



Notice that the '**book seats**' button is *greyed out* until the panel is closed by the user.

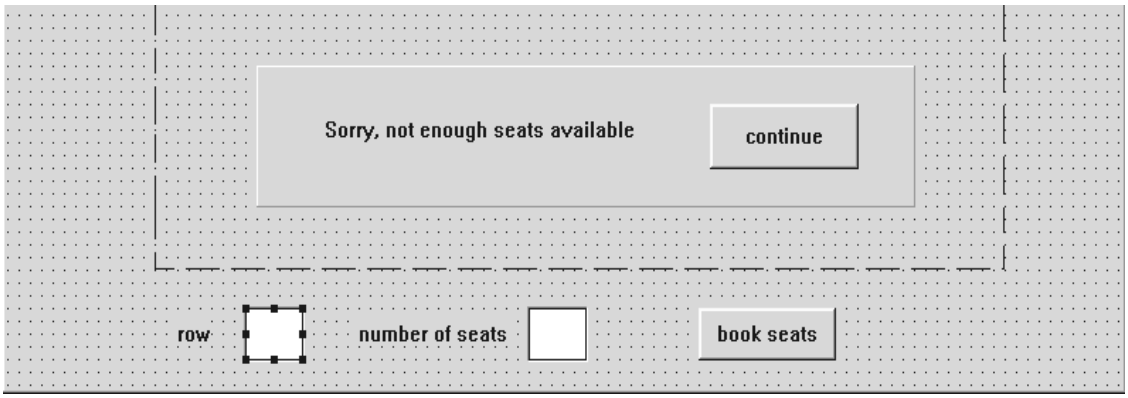
The final step is to increase the number of rows of seats to 8 for the whole theatre. Go first to the ***var*** heading and change the **row** variable to an array:

```
var
  Form1: TForm1;
  row: array[1..8] of TRow;
```

Modify the FormCreate procedure by adding a loop to initialise all 8 rows of seats:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  m,n:integer;
begin
  with image1.canvas do
  begin
      ......
    textout(280,30,'stage');
  end;
  for m:=1 to 8 do
  begin
    row[m]:=TRow.create;
    row[m].rownumber:=m;
    for n:=1 to 12 do
    begin
      row[m].seat[n]:=TSeat.create;
      row[m].seat[n].seatnumber:=n;
      row[m].seat[n].rownumber:=m;
      row[m].seat[n].booked:=false;
    end;
    row[m].display;
  end;
end;
```

To make bookings the user will need to give the number of the row, as well as the number of seats required. Go to the Form grid and add another ***edit box*** for this:



Place a ***label*** alongside with the caption '**row** '.

Double-click the edit box to create an event handler and add the lines:

```
procedure TForm1.Edit2Change(Sender: TObject);
begin
  if edit2.text='' then
    r:=0
  else
    r:=strtoint(edit2.text);
end;
```

Add the variable ***r*** to the *Public declarations* section:

```
            r:integer;
```

Make modifications to the '**book seats**' button click procedure as shown. We have introduced an error trapping line to ensure that a valid row number in the range 1-8 was selected. Each occurrence of ***row***must be replaced by ***row[r]***:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  count,i:integer;
begin
  if (r>=1) and (r<=8) then
  begin
    count:=0;
    for i:=1 to 12 do
    begin
      if row[r].seat[i].booked=false then
        count:=count+1;
    end;
    if count>=wanted then
    begin
      count:=0;
      i:=0;
      repeat
        i:=i+1;
        if row[r].seat[i].booked=false then
        begin
          row[r].seat[i].booked:=true;
          count:=count+1;
        end;
      until count=wanted;
      row[r].display;
    end
    else
    begin
        ........
    end;
  end;
end;
```

Compile and run the completed program. Carry out testing to check that bookings can be made for any of the 8 rows: