Content
This chapter explains the main concepts of objects-orientation and their adaption in ABAP. Furthermore the concept of Business Server Pages (BSPs) is explained and as well as related topics like the use of BAPIs, HTMLB and the Model-View-Controller pattern.

Prerequisites
You should be familiar with ABAP and the navigation in the SAP system.

Motivation
This chapter explains how to create simple programs in ABAP Objects. Furthermore it shows how to maintain and execute BSP applications, how to use a BAPI within a BSP and how to design BSPs using HTMLB. Above that, this chapter explains how to adapt the Model-View-Controller pattern for BSPs.

Lecture notes
The fundamental understanding of the ABAP development in the SAP system is a prerequisite for the students. Students can go on with their account from chapter 1.

- **Product:** All
- **Level:** Beginner
- **Focus:** Programming
- **Version:** 1.0
- **Author:** UCC Technische Universität München
Task 1: Login into the SAP system

Short description: Use SAPGui to login into the SAP system with your username and password.

Start the SAPGui and login into the development system using the provided account and password. Please refer to chapter 1 for your username and your password.

Task 2: Working with ABAP Objects

Short description: Use the concept of ABAP Object within an ABAP program.

Please start the Object Navigator from the SAP Easy Access Menu by using the following path:

**Tools • ABAP Workbench • Overview • Object Navigator.**

You may also use the transaction code **SE80** for direct access.

Create a new program named ‘ZY_##_OBJECTS’. Do not use a TOP-Include and declare the program as test program.

You will create a new local class with the purpose of saving airlines. This class will be called in the program and several new instances of this class will be created.

First add command ‘INCLUDE ZY_##_CLASS_INCLUDE.’ in your program and create the corresponding include program by the use of forward navigation (double click). The newly created include program will include definition and implementation of the local class **LCL_AIRPLANE**.

Please start with the class declaration. You require a method **set_attributes** and a method **get_attributes** with public access. Method **set_attribute** includes an importing parameter **im_name** for the airplane name and importing parameter **im_planetype** for the airplane type. Both attributes have to be defined in the private section as attributes. The declaration of the local class is shown in Hiba! A hivatkozási forrás nem található.

```abap
CLASS lcl_airplane DEFINITION.
PUBLIC SECTION.
METHODS set_attribute IMPORTING
      im_name TYPE string
      im_planetype TYPE string,
      get_attribute.
PRIVATE SECTION.
DATA: name TYPE string,
      planetype TYPE string.
ENDCLASS. "lcl_airplane DEFINITION"
```

Method **set_attributes** should provide functionality to set both private attributes **name** and **planetype** according to the importing parameters. Method **get_attributes** should provide functionality to output the attribute values for a certain object with help of the WRITE-command. The implementation of both methods is shown here:
Please return to the main program in order to implement the use of the class. You will now create several airplanes using a reference and save the references on these objects in an internal table, in order to access these objects again at a later date. The reference \texttt{r\_plane} will use a new type definition: It will refer to the local class \texttt{LCL\_AIRPLANE} by the use of \texttt{TYPE REF TO}. In the same way internal table \texttt{it\_plane\_list} is defined as table of references on this local class. The data declaration of program \texttt{ZY\_##\_OBJECTS} therefore looks as shown in the figure bellow:

```
REPORT  xx_00_objects.

INCLUDE xx_00_class_include.

DATA: r_plane TYPE REF TO lcl_airplane,
     it_plane_list TYPE TABLE OF REF TO lcl_airplane.
```

Now start the block \texttt{START-OF-SELECTION} by creating an object with command \texttt{CREATE OBJECT} that is referenced by variable \texttt{r\_plane}. Call method \texttt{set\_attribute} using this reference and assign name and plantype to the new object using the corresponding parameters of the method. Append the reference to table \texttt{it\_plane\_list} using the command \texttt{APPEND}. By appending the reference the address of the created object is saved in the table and the reference \texttt{r\_plane} itself can be used for other objects. Now create a new airplane and append this reference to the internal tables as well (compare Hiba! A hivatkozási forrás nem található.).

```
CREATE OBJECT r_plane.
    r_plane->set_attribute( im_name  = 'Hamburg' ,
                            im_plantype = 'Boeing 737' ).
    APPEND r_plane TO it_plane_list.
```

```
CREATE OBJECT r_plane.
    r_plane->set_attribute( im_name  = 'Munich' ,
                            im_plantype = 'Airbus 380' ).
    APPEND r_plane TO it_plane_list.
```

Finally you will loop at the internal table and display data for each airplane on the screen using method \texttt{get\_attribute}.

```
REPORT  xx_00_objects.

INCLUDE xx_00_class_include.

DATA: r_plane TYPE REF TO lcl_airplane,
     it_plane_list TYPE TABLE OF REF TO lcl_airplane.
```

Now start the block \texttt{START-OF-SELECTION} by creating an object with command \texttt{CREATE OBJECT} that is referenced by variable \texttt{r\_plane}. Call method \texttt{set\_attribute} using this reference and assign name and plantype to the new object using the corresponding parameters of the method. Append the reference to table \texttt{it\_plane\_list} using the command \texttt{APPEND}. By appending the reference the address of the created object is saved in the table and the reference \texttt{r\_plane} itself can be used for other objects. Now create a new airplane and append this reference to the internal tables as well (compare Hiba! A hivatkozási forrás nem található.).

```
CREATE OBJECT r_plane.
    r_plane->set_attribute( im_name  = 'Hamburg' ,
                            im_plantype = 'Boeing 737' ).
    APPEND r_plane TO it_plane_list.
```

```
CREATE OBJECT r_plane.
    r_plane->set_attribute( im_name  = 'Munich' ,
                            im_plantype = 'Airbus 380' ).
    APPEND r_plane TO it_plane_list.
```

Finally you will loop at the internal table and display data for each airplane on the screen using method \texttt{get\_attribute}.
Please save, check and activate your new program. Finally test it. You should see the attributes of both created objects.

**Task 3: Create a simple Business Server Page (BSP)**

**Short description:** Create a new BSP Application.

Please start the Object Navigator from the SAP Easy Access Menu by using the following path:

**Tools · ABAP Workbench · Overview · Object Navigator.**

You may also use the transaction code **SE80** for direct access.

Open your previously created package by selecting the entry 'Package' from the drop down list and type in the name of your package. Please hit 'Enter'.

Create a new BSP application 'ZY_##_BSP' using the path **Create · BSP Library · BSP Application** of the context menu of your package. In the pop-up type in the name of your BSP application and maintain the short text.

Assign all newly created objects to your package and transport request. Save your new BSP application. In your navigation tree a new node **BSP Library** is shown now. You can find your BSP Application within the sub node **BSP Applications.**
Create a new page named display.htm using the context menu of this application. In the area Page Type please activate radio button ‘Page with flow logic’.

Switch to the tab ‘Layout’ in order to display your new page. Delete the already generated source code completely. Now enter the server side language that will be used for the BSP; in this case the server side language is ABAP. Create with common HTML-Code a new page and enter a title of your choice. Insert a data declaration for an internal table and a work area referring to table ‘SPFLI’ in ABAP. The declaration should be embraced by the tags \(<%\) and \(\%>\). Furthermore add a SELECT-statement that moves all flight connections of table SPFLI into internal table it_flights:

```html
<%@page language="abap" %>
<html>
<head>
  <title>My first BSP page</title>
</head>
<body>
  data : it_flights type table of spflti,
    wa_flights type spflti.
  select * from spflti into table it_flights.
</body>
<html>
```

Next create a HTML table. The first row of the table includes the headlines for Airline, Flight Connection, Departure airport and time as well as Arrival airport and time. Further rows of the table will be created using a LOOP-statement on the internal table ‘it_flights’.
Save, check and activate the application. As soon as you start testing the page a new web browser session is opened and you have to log in with your user ID. The result should look similar to this:

### Display all data of table SPFLI

<table>
<thead>
<tr>
<th>Airline</th>
<th>Connection</th>
<th>Departure airport</th>
<th>Arrival airport</th>
<th>Departure time</th>
<th>Arrival time</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>0817</td>
<td>JFK</td>
<td>SFO</td>
<td>110000</td>
<td>140000</td>
</tr>
<tr>
<td>AA</td>
<td>0864</td>
<td>SFO</td>
<td>JFK</td>
<td>090000</td>
<td>172000</td>
</tr>
<tr>
<td>AZ</td>
<td>0555</td>
<td>FCO</td>
<td>FRA</td>
<td>190000</td>
<td>210500</td>
</tr>
<tr>
<td>AZ</td>
<td>0780</td>
<td>FCO</td>
<td>TYO</td>
<td>120000</td>
<td>085500</td>
</tr>
<tr>
<td>AZ</td>
<td>0789</td>
<td>TYO</td>
<td>FCO</td>
<td>114500</td>
<td>192500</td>
</tr>
<tr>
<td>AZ</td>
<td>0790</td>
<td>FCO</td>
<td>KIX</td>
<td>103500</td>
<td>031000</td>
</tr>
</tbody>
</table>

**Task 4: Use of BAPIs in Business Server Page (BSP)**

**Short description:** Use a BAPI in your BSP application to access data from another SAP System.

Please start the Object Navigator from the SAP Easy Access Menu by using the following path:

**Tools • ABAP Workbench • Overview • Object Navigator.**

You may also use the transaction code **SE80** for direct access.

For this task use again your BSP Application **ZY_##_BSP**. Create a new page named **call_bapi.htm** using the context menu of your application. In the area **Page Type** please activate option **Page with flow logic.**
After confirmation of the dialog screen you are directed to the 'Layout' tab of the new BSP page. First of all please delete some code lines so your source code looks similar to this:

```html
<!-- delete some code -->
```

As you can see we are already using HTMLB for this page. But as there have been no HTMLB-specific elements in use yet, we do not regard HTMLB any further in this step.

Now you want to include the BAPI call. For this step you need to find out, which import and export parameters are required for the BAPI call. Therefore switch to the BAPI Browser using the menu path **BAPI Browser**.

In the BAPI browser scroll down till you find node ‘G51’. These shown nodes are so called RFC connections (remote function calls), which are used by the SAP system to transfer data between SAP systems and also to transfer data to itself. Open the node ‘G51’ please (this may take a little while) and scroll down till you find the BAPI ‘BAPI_USER_GET_DETAIL’. The BAPI is located nearly at the end of the list.
By double clicking on it the SAP system opens the description and the call examples for the BAPI. You want to use the BAPI to get answers about two questions:

- When was the user ‘xyz’ last changed?
- Is the user account locked?

To call the BAPI in your BSP you have to perform a ‘CALL FUNCTION’ and to export the RFC connection as well as to export parameters e.g. a username. The incoming data from the function will be stored in temporary variables. The following snippet shows you the ABAP code of your page. Place the ABAP code before the HTMLB code:

```abap
1. <page language='abap' %>
2. <extension name='htmlb' prefix='htmlb' %>
3. <
4. * Define variables
5. data: tmp_islocked TYPE DAPILOCKS, tmp_moddate TYPE DAPILOGDAT, tmp[20] TYPE c, mod TYPE date.
6. *
7. * Perform BAPI call
8. call function 'BAPI_USER_GETDETAIL' 
9. destination 'CS1'.
10. importing islocked = 'islocked-is'
11. importing lastmodified = 'tmp_moddate'.
12. *
13. * Convert received data
14. if tmp_islocked = '0000'.
15.     mod = 'user is unlocked'.
16. else.
17.     mod = 'user is locked'.
18. endif.
19. endif.
20. end.
```

The ABAP code does the following: at first you define some temporary variables, then call the BAPI by using the call function instruction and then convert the received results into a useable format (this is done by the if instruction). Now you have to define some text field to display the results. The entire source code of your new program should look similar to this:
Save, check and activate all of your objects. Test the little example. The results should be similar to this:

Result

You may want to test some other features for the text fields and some other BAPIs. Please feel free to do so.

**Task 5: Take advantage of HTMLB**

**Short description:** Use HTMLB to display the result of a SQL statement using one HTMLB instruction.

In task 4 you used the usual HTML for displaying some data from the flight example. As you remember building the HTML table was very time consuming and fault-prone. Now you want to do this once again, but this time using HTMLB.

Please start the Object Navigator from the SAP Easy Access Menu by using the following path:

Tools • ABAP Workbench • Overview • Object Navigator.
You may also use the transaction code **SE80** for direct access.

For this task use again your BSP Application ‘**ZY_##_BSP**’. Create a new page named ‘htmlb_connections.htm’ using the context menu of your application. In the area **Page Type** please activate option **Page with flow logic**.

In the generated code of the new BSP, please delete all code lines between the `<htmlb:form>` tags. Now maintain the necessary ABAP code fragments to read the connection details from the database and make them available in the BSP. Refer to the following illustration if you encounter any problems:

In the next step you want to display all data. Instead of building a table manually you will take advantage of HTMLB to display all the data. This can be achieved by using the `<htmlb:tableview>` tag. Insert the following code fragments:
Now save, check and activate your new BSP. By testing it you will see a big difference between the old fashioned HTML table and the new HTMLB table view, which displays the data and gives you additional functionalities.

### Task 6: Use the MVC pattern

**Short description:** Use the MVC pattern to build a new web application, which uses separated layers for presentation and flow logic.

Please start the Object Navigator from the SAP Easy Access Menu by using the following path:

**Tools • ABAP Workbench • Overview • Object Navigator**

You may also use the transaction code SE80 for direct access.

For this task you have to create a new BSP application ‘ZY_##_MVC’. Please do so by using the context menu of your package. Maintain the short description of the new application and go on then. Assign the new application to your existing package and assign everything to your existing transport request.

Now create a new controller named ‘zy_##_mvc_con.do’ by using the context menu of your BSP application.
Type in the name of your new controller and maintain the short description.

As you created a controller for your MVC application, the controller does not contain any flow logic or any layout. You have to assign the controller to a controller class now. Because you have not created a controller class until now, you enter the name of your future class 'ZYCL_##_MVC'. Save your controller and double click on the class name. You will be asked if you want to create the new class. Please confirm and assign the new class to your package and your transport request.

As you can see your newly created class does automatically own a lot of methods. In fact these methods are inherited from the SAP class 'CL_BSP_CONTROLLER2'. The SAP system has automatically chosen this class as you created a new class for a new controller. Your new class also contains a method called 'DO_REQUEST'. This method is executed whenever the controller receives a request. You will use this method to implement your flow logic. As your class inherited the method from the SAP standard class, you are not allowed to change the method, but you are allowed to redefine it. And this is what you will do now. So please click on the method and redefine it. Therefore place your cursor on the method and then press the 'Redefine' button.
Please insert the following code fragment:

```abap
method DO_REQUEST.
    data: myview type ref to IF_BSP_PAGE.
    myview = create_view(view_name = 'zy_99_mvc_view.html').
    myview->set_attribute(name = 'name' value = sy-uname).
    call_view( myview ).
endmethod.
```

The code does the following: at first it defines a new view named ‘myview’, which is derived from the class ‘IF_BSP_PAGE’. After that the view is created via ‘create_view’. As you can see you already transfer the name of a BSP page although you have not created it till now. It is essential to use the same name for your view later on. After the creation you want to transfer some attributes to the view, which should be displayed in the view later on. To do so we use the ‘set_attribute’ method of ‘myview’. The last instruction calls the view.

In the next step you want to create the view, which you already used in your controller. Please create a new page but this time you choose the radio button ‘View’. It is necessary to choose the same name you already used in the controller!

The idea of the MVC application is to display the username of the user that runs the application. Therefore the view has to display the username now, which is determined by the controller. By using the attributes in the controller you transfer the username from the controller to the view. Please change the pregenerated source code slightly:

```html
<%@page language="abap"%>
<%@extension name="html" prefix="html"%>

<html:content design="design03">
    <html:page title = "View for MVC application ">
        <html:form>
            <html:textView text = "You are logged in!"
                            design = "EMPHASIZED" />
            <html:textView text = "<i>" name = "%" design = "EMPHASIZED" />
        </html:form>
    </html:page>
</html:content>
```
In the last step you have to define the attribute which should be transferred from the controller to your view. This can be done in the tab ‘Page Attributes’. Please define the attribute ‘name’ and use the type ‘SY-UNAME’.

Save, check and activate all of your objects and test your new MVC application by testing the controller. You will see that the view displays the current username although it does not contain any flow logic.