PROJECT REPORT

ON

"Project Development using ABAP in SAP-INDIA"

submitted in the partial fulfillment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Submitted To:
Ms. S. Adhirai
Assistant Professor
Head, Deptt. of CSE

Submitted By:
Rohit Sharma
Univ RollNo : 9148585
CSE 8th SEM

Under the Guidance of:
Mr. Ajay Dureja,
Assistant Professor
Deputy Head, CSE Department

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
PDM COLLEGE OF ENGINEERING, BAHADURGARH

MAY, 2019

DECLARATION BY THE CANDIDATE

I, **Rohit Sharma,** hereby declare that the project work entitled **"Project Development using ABAP in SAP-INDIA"** is an authenticated work carried out by me at **PDM College of Engineering** for the partial fulfillment of the award of the degree of Bachelor of Technology in Computer Science & Engineering submitted at, PDM College of Engineering, Bahadurgarh, Haryana. This work has not been submitted for similar purpose anywhere else.

Date:
Place: Haryana

Rohit Sharma
Univ. Roll-9148585
B.Tech CSE 8th Sem

CERTIFICATE

This is to certify that the Project Work entitled **"Project Development using ABAP in SAP-INDIA"**, which is being submitted by **Rohit Sharma** and **Roll No. 9148585** in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science & Engineering submitted at, PDM College of Engineering, Bahadurgarh is an authentic record of his work carried out under my supervision.

The matter presented in this project work has not been submitted for the award of anyother degree of this or any other university.

Project Guide
Mr Ajay Dureja
Assistant Professor, CSE
Sector 3-A, Sarai Aurangabad
Bahadurgarh

Internal Guide
Ms Suruchi Nehra
Assistant Professor, CSE
Sector 3-A, Sarai Aurangabad,
Bahadurgarh

ACKNOWLEDGEMENT

I would specially like to thanks my worthy guide **Mr. Ajay Dureja**, **Assistant Professor, CSE Department, PDM College of Engineering** who supervised me to complete this project. His technical advice, ideas and constructive criticism contributed to the success of this report. She suggested me many ideas and solved my puzzles when I was in need. his motivation and help has been of great inspiration to me.

I would also like to thanks my respected HOD (CSE) and faculty members of CSE department for providing me the opportunities, support and the necessary help to complete this project work.

Rohit Sharma
Univ. Roll-9148585
B.Tech CSE 8th Sem

ABSTRACT

SAP stands for SYSTEMS, APPLICATION AND PRODUCTS in data processing. SAP in data processing is business software that integrates all applications running in an organisation. These applications represent various modules on the basis of business area such as finance, sales and distribution and are jointly executed to accomplish to achieve overall business logics.

ABAP (Advanced Business Application Programming), is a fourth-generation programming language, used for development and customization purposes in the SAP software.

1. Reports
2. Module Pool Programming
3. Interfaces
4. Forms
5. Data conversions
6. User Exits & BADI

All of R/3's applications and even parts of its basis system were developed in ABAP.

ABAP is an event-driven programming language. User actions and system events control the execution of an application.

ABAP is also called ABAP/4. The "4" in ABAP/4 stands for "Fourth Generation Language" or 4GL.

LIST OF FIGURES

CONTENTS

TITLE **PAGE**

CHAPTER 2

INTRODUTION TO ABAP

CHAPTER 3

ABAP DICTONARY

CHAPTER 4

ABAP VIEWS

CHAPTER 8

WEB DYNPRO

CHAPTER 9

PROGRAMS

CHAPTER 1

INTRODUCTION TO SAP

SAP stands for **SYSTEMS, APPLICATION AND PRODUCTS** in data processing. SAP in data processing is business software that integrates all applications running in an organisation. These applications represent various modules on the basis of business area such as finance, sales and distribution and are jointly executed to accomplish to achieve overall business logics.

1.1 HISTORY

SAP was developed by SAP AG, Germany, The basic ideas behind developing SAP was the need of standard application software that helps in real-time business processing.

The development process began in 1972 with five IBM employees, Dietmar Hopp , Hanserner Hector , Hasso Plattner , Klaus and Claus in Germany.

SAP is a market leader in providing **ERP** (Enterprise Resource and Planning) solutions and services. In this chapter, we will try to understand more on ERP and where it should be used. In addition, we will learn the implementation techniques of ERP along with the ERP packages available in the market.

1.2 WHAT IS ERP?

Enterprise Resource Planning (ERP) is a software that is built to organizations belonging to different industrial sectors, regardless of their size and strength.

The ERP package is designed to support and integrate almost every functional area of a business process such as procurement of goods and services, sale and distribution, finance, accountings, human resource, manufacturing, production planning, logistics & warehouse management.

Fig. 1.1 ERP Process.

1.2.1 Business Process Integration

Every business, regardless of the industry they belong to, require connected systems with efficient information flow from one business process to another. Business Process Integration (BPI) plays an important role in overcoming integrating challenges that allows organizations to connect systems internally and externally.

Business Process Integration (BPI) allows −

1. automation of business processes,
2. integration of systems and services,
3. secure sharing of data across numerous applications, and
4. automation of management, operational, and supporting process.

The following illustration shows an overview of various business processes running in an enterprise and how they are integrated.

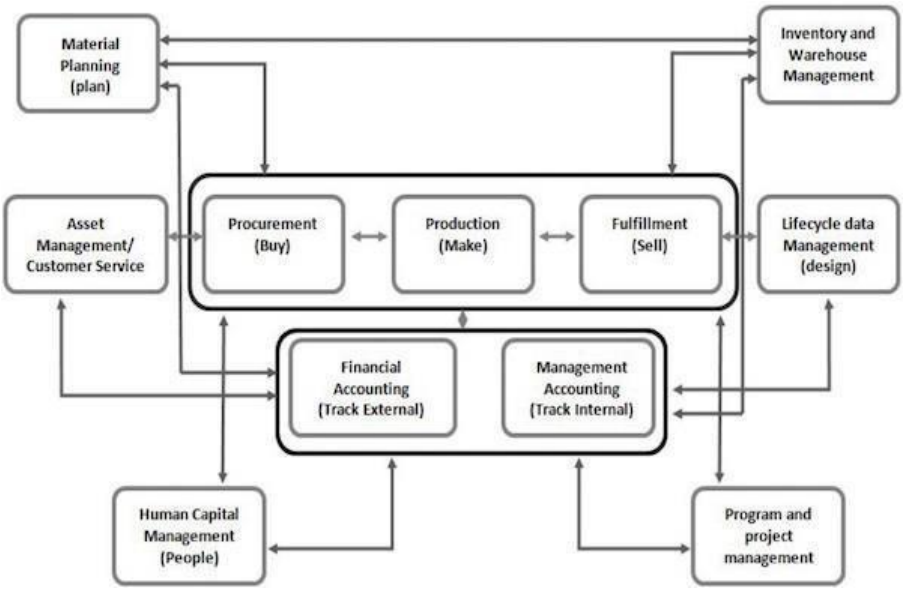


Fig. 1.2 Business Process Integration.

1.2.2 Evolution of ERP

During early phases of development, integrated solutions were designed for particular process areas such as −

1. Material Management − the integrated system was known as Material Requirement Planning (MRP)
2. Manufacturing − the integrated system was known as Manufacturing Resource Planning

However none of the integrated systems came with a complete solution for an organization covering major business process areas. In early 1990's, the Gartner Group first used the acronym **ERP**. By mid–1990's, ERP systems addressed all the core enterprise functions.

In the early stages, most of the ERP solutions were focused on automating *back office functions* that were not directly affecting customers or general public. Later, *front office functions* such as customer relationship management and e–business systems were integrated.

1.2.3 Functions of ERP

An ERP system typically performs the following functions −

1. Supports the integrated business process inside the organization.

2. Improves capital planning and helps in executing organizational plans and strategies.
3. Helps speed up the decision-making process over the analysis of accurate data.
4. Helps extend the business network to wider domains, expanding the products and services to reach more customers, suppliers, and partners.
5. Identifies operational risks to improve governance.
6. Provides protection against organizational data breaches and security threats to leakage of information.
7. Makes the organization adaptable to the rapid changes in the business process according to the needs.
8. Gives long-term profit by providing means to increase the customer base.

1.2.4 Functional Areas

ERP is a business management software is usually a suite of integrated applications that a company can use to collect, store, manage, and interpret data from many functional areas including −

a) **Financial Accounting** − Deals with financial transactions and data.

b) **Human Resource** − Deals with information related to employee of an organization.

c) **Customer Relationship Management** − Deals with capturing and managing customer's relationship, facilitating the use of customer experience to evaluate the knowledge database.

d) **Sales and Distribution** − Deals with order placement, delivery, shipment and invoicing.

e) **Logistics and Warehouse Management** − Deals with storage of products and shipment.

f) **Manufacturing and Material Management** − Deals with the production and production planning activities.

g) **Supply Change Management** − Deals with the movement of products, storing, managing, and controlling supplies.

h) **Business Intelligence** − Analyzes data and converts the same to information.

1.2.5 Advantages of ERP

By integrating the business processes, the ERP offers the following advantages −

- Saves time and expenses.
- Allows faster decision-making by the management, utilizing the data and reporting tools designed in the systems.
- Single data source and sharing of data among all the units of an organization.
- Helps in tracking every transaction that takes place in an organization, from starting till end.
- Supplies real-time information whenever required.
- Provides synchronized information transfer in between different functional areas such as sales, marketing, finance, manufacturing, human resource, logistics, etc.

1.2.6 Disadvantages of ERP

It is not always easy to incorporate ERP in an organization. ERP suffers from the following drawbacks −

- Sometimes business processes critical to an organization are to be re-engineered to align them with an ERP solution.
- Cost of complex integration can be very high.
- Switching from one ERP solution to another increases the implementation cost even further.
- End-users are to be trained for their daily operations.
- Customization is not preferred.

1.3 MODULES IN SAP

SAP solutions include a number of functional modules, which support transactions to execute key business processes, such as −

1. Financial Accounting (FI)
2. Financial Supply Chain Management (FSCM)

3. Controlling (CO)
4. Materials Management (MM)
5. Sales and Distribution (SD)
6. Logistics Execution (LE)
7. Production Planning (PP)
8. Quality Management (QM)
9. Plant Maintenance (PM)
10. Project System (PS)
11. Human Resources (HR)

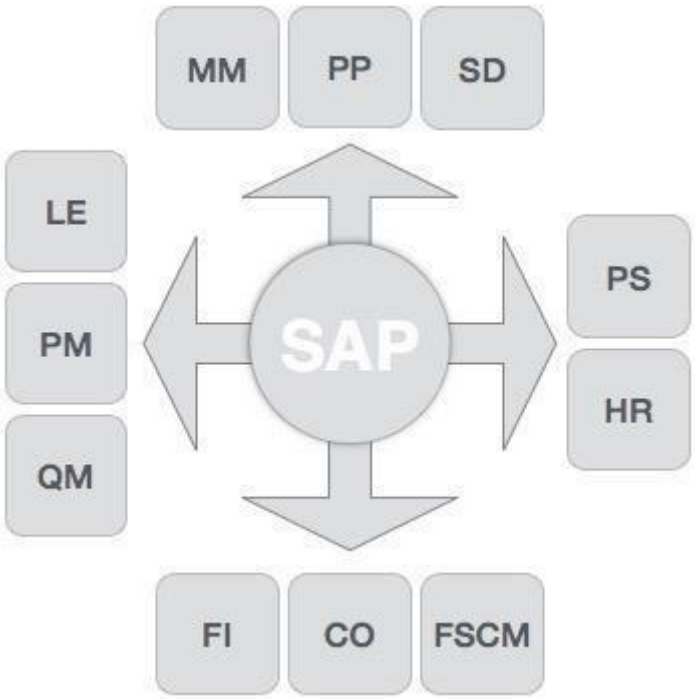


Fig. 1.3 Modules in SAP.

1.3.1 Finance and Controlling (FICO)

SAP FICO is a combination of two ERP modules, i.e., Finance Accounting (FI) and Controlling (CO). Under Finance in SAP and at an enterprise level, the following modules take part −

- FI − Finance
- CO − Controlling

- IM − Investment Management
- TR − Treasury
- EC − Enterprise Controlling

SAP FI (Financial Accounting) is accountable for tracking the flow of financial data across the organization in a controlled manner and integrating all the information for effective strategic decision-making.

1.3.1.1 Activities Involved in SAP FI

- Creation of Organizational Structure (Defining Company, Company Codes, business Areas, Functional Areas, Credit Control, Assignment of Company Codes to Credit Controls)
- Financial Accounting Global Settings (Maintenance of Fiscal Year, Posting Periods, defining Document types, posting keys, Number ranges for documents)
- General Ledger Accounting (Creation of Chart of Accounts, Account groups, defining data transfer rules, creation of General Ledger Account)
- Tax Configuration & Creation and Maintenance of House of Banks
- Account Payables (Creation of Vendor Master data and vendor-related finance attributes like account groups and payment terms)
- Account Receivables (Creation of Customer Master data and customer-related finance attributes like account groups and payment terms
- Asset Accounting
- Integration with SD and MM

SAP CO (Controlling) module facilitates coordinating, monitoring, and optimizing all the processes in an organization. It controls the business flow in an organization. This module helps in analyzing the actual figures with the planned data and in planning business strategies.

Two kinds of elements are managed in CO −

1. Cost elements
2. Revenue elements

These elements are stored in the FI module.

1.3.1.2 Activities Involved in SAP CO

a) Cost Element Accounting (Overview of the costs and revenues that occur in an organization)

b) Cost Center Accounting

c) Activity-Based-Accounting (Analyzes cross-departmental business processes)

d) Internal Orders

e) Product Cost Controlling (Calculates the costs that occur during the manufacture of a product or provision of a service)

f) Profitability Analysis (Analyzes the profit or loss of an organization by individual market segments)

g) Profit Center Accounting (Evaluates the profit or loss of individual, independent areas within an organization)

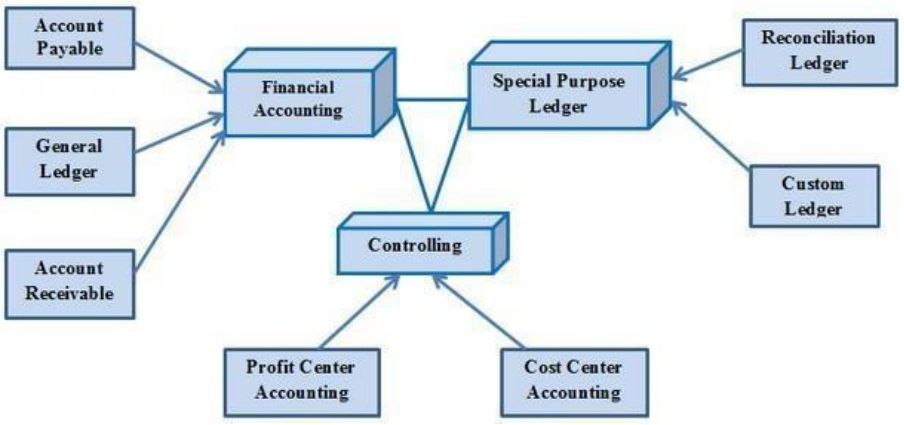


Fig. 1.4 Activities involved in SAP CO.

1.3.2 Sales & Distribution Management (SD)

SAP SD is one of the most important modules in SAP. It has a high level of integration complexity. SAP SD is used by organizations to support sales and distribution activities of products and services, starting from enquiry to order and then ending with delivery.

SAP SD can monitor a plethora of activities that take place in an organization such as products enquires, quotation (pre-sales activities), placing order, pricing, scheduling

deliveries (sales activity), picking, packing, goods issue, shipment of products to customers, delivery of products and billings.

In all these processes, multiple modules are involved such as FI (Finance Accounting), CO (Controlling), MM (Material Management), PP (Production Planning), LE (Logistics Execution), etc., which shows the complexity of the integration involved.

1.3.2.1 Activities Involved in SAP SD

a) Setting up Organization Structure (creation of new company, company codes, sales organization, distribution channels, divisions, business area, plants, sales area, maintaining sales offices, storage location)

b) Assigning Organizational Units (Assignment of individual components created in the above activities with each other according to design like company code to company, sales organization to company code, distribution channel to sales organization, etc.)

c) Defining Pricing Components (Defining condition tables, condition types, condition sequences)

d) Setting up sales document types, billing types, and tax-related components

e) Setting up Customer master data records and configuration

1.3.3 Material Management(MM)

Material Management deals with movement of materials via other modules like logistics, supply chain management, sales and delivery, warehouse management, production and planning.

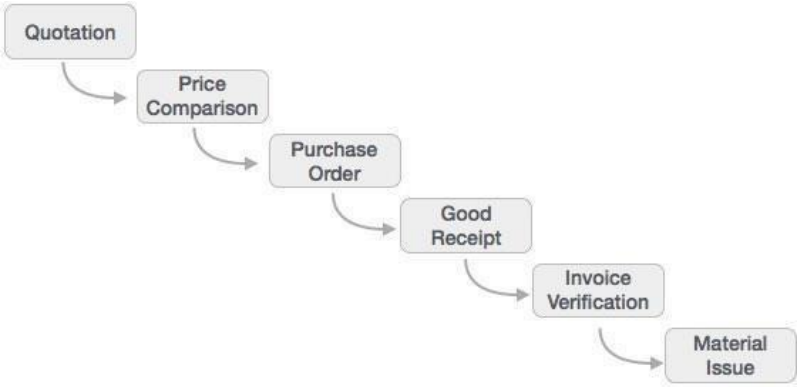


Fig. 1.5 Activities involved in Material Management.

1.3.4 Logistic Execution(LE)

Logistic Execution can be divided into two sub-modules, i.e., shipment of goods (purchase to procurement process) and warehouse management (storage of goods). These two modules are integrated with sale and distribution, material management, and production and planning.

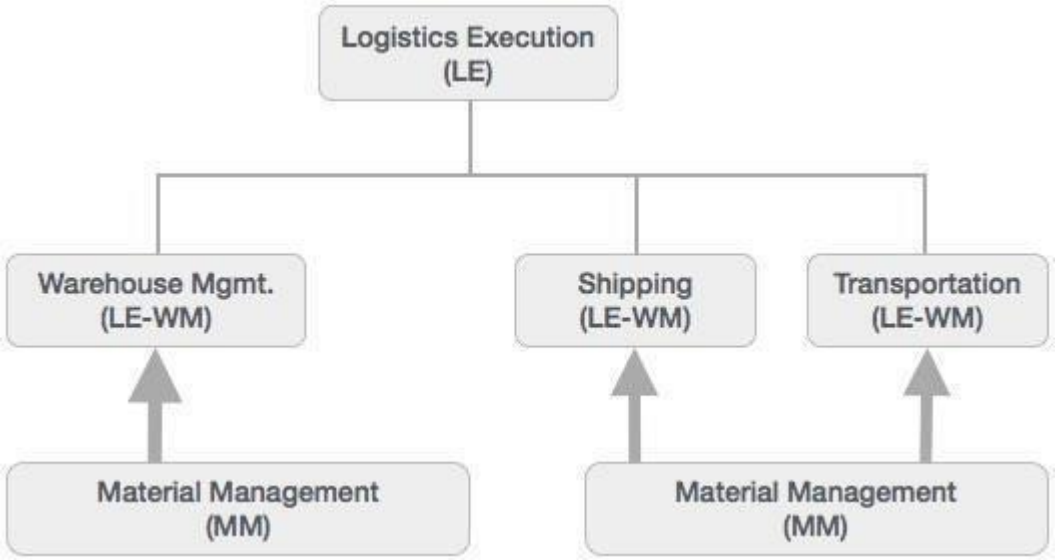


Fig. 1.6 Activities involved in Logistic Execution.

1.3.5 Supplier Relationship Management(SRM)

As the name SRM suggests, this module deals with the effective and efficient transition of products and services between an organization and its suppliers. The main process covered in this section is procurement of products like direct materials, indirect materials, and services. This module can effectively integrate with planning, accounting, and inventory system.

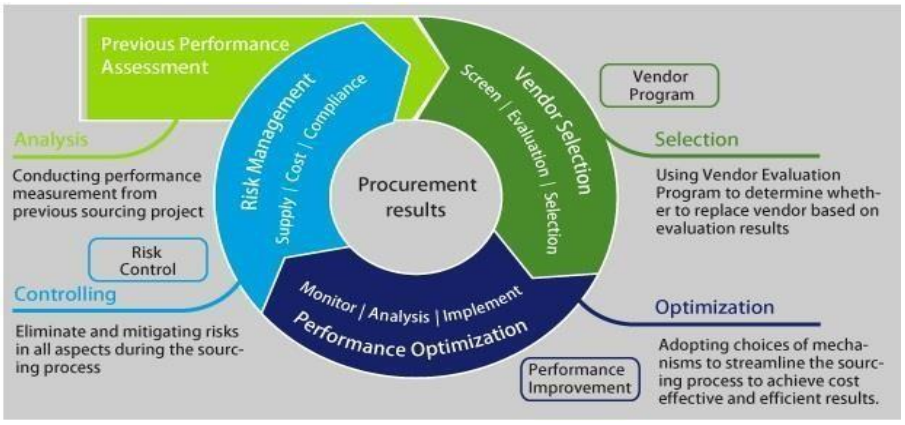


Fig. 1.7 Activities involved in SRM.

1.3.6 End-to-End Procurement Cycle

Procurement process with SAP Enterprise Buyer comprises of the following major steps –

a) Shopping Carts
b) Approval of Shopping Cart
c) Sourcing of Requirements
d) Purchase Orders
e) Purchase Order Approval
f) Confirm Goods/Services
g) Confirmation Approval
h) Process Invoice
i) Invoice Approval

1.3.7 Customer Relationship Management (CRM)

CRM deals with end-to-end customer related processes. CRM is designed to centralize the data related to all the customers associated with an organization. It helps an organization −

a) Maintain its sales, services, and build marketing strategies according the market demand and customer data analysis.

b) Remain focused on its customers and via information analysis, help the business to know more about its customers.

c) Improve sales and services and building better relationships with customers.

Fig. 1.8 Activities involved in CRM.

1.3.8 Human Resource(HR)

The most important objective of master data administration in Human Resources is to enter employee-related data for administrative, time-recording, and payroll purposes.

A new employee can be hired without using Recruitment. Instead you can hire someone by running a personnel action in Personnel Administration, thereby creating the necessary data for the employee to be hired.

Employee data must be kept current. After an employee is hired, circumstances can always arise which necessitate either the entry of new data or the correction of current data. For instance −

a) An employee moves to his or her new address must be stored in the system.

b) An employee gets a pay hike at the start of the year. The new salary must be stored for the relevant date.

c) An employee changes jobs within the organization. His or her organizational assignment, working time, and salary also change.

d) Data can be stored for the past, present, or future.

Entering payroll-relevant data in the past triggers retroactive accounting.

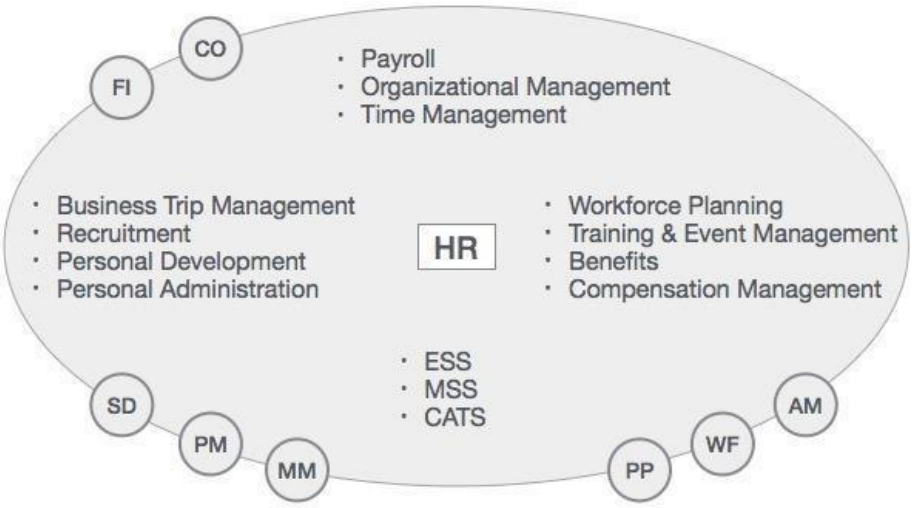


Fig. 1.9 Activities involved in HR.

The HR module is comprised of major areas of functionality known as sub-modules. The HR module is a true demonstration of the strength of the SAP product in Enterprise Resource Planning.

The HR system has very strong integration points (where data is passed back and forth without human intervention) with just about all of the other SAP modules. In addition, there is very tight integration amongst the HR sub-modules.

The above illustration highlights some of the basic SAP HR terms as listed below.

1.4 ABAP(Advanced Business Application Programming)

ABAP is a programming language that runs in the SAP ABAP runtime environment, created and used by SAP for the development of application programs including:

7. Reports
8. Module Pool Programming
9. Interfaces
10. Forms
11. Data conversions
12. User Exits & BADI

All of R/3's applications and even parts of its basis system were developed in ABAP.

ABAP is an event-driven programming language. User actions and system events control the execution of an application.

ABAP is also called ABAP/4. The "4" in ABAP/4 stands for "Fourth Generation Language" or 4GL.

1.4.1 Benefits brought by the SAP

- Faster time to serve customers, external and internal.
- Smaller rework-integration between people and information.
- Point of contact for customers
- Easier to measure results
- Optimization of costs.

1.4.2 SAP ARCHITECTURE

- SAP system has a 3-tier Architecture
- Presentation Layer (Windows based)
- Application Layer (Windows Server/UNIX)
- Database Server Database Layer (Windows Server/UNIX)

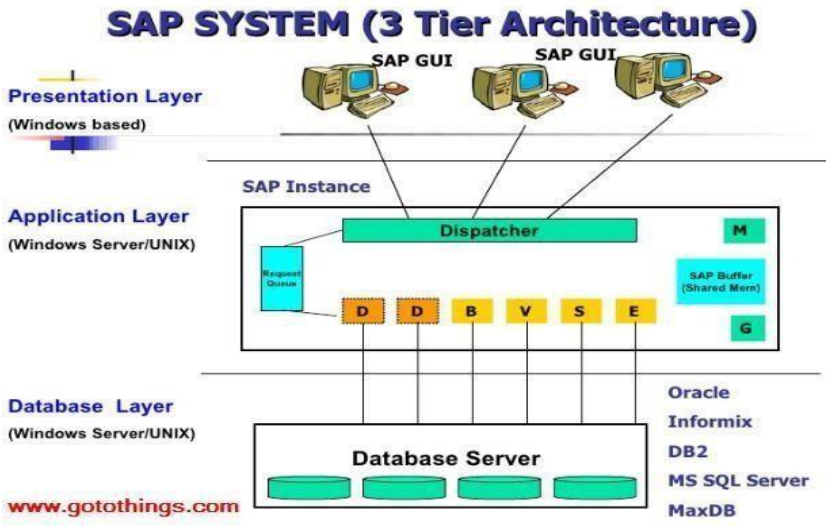


Fig. 1.10 SAP Architecture.

1.4.2.1 Presentation Servers

Presentation servers contain systems capable of providing a graphical interface.

- Presentation Layer is also known as client Layer
- Presentation Layer is a user interaction
- In SAP-User interaction purpose we use GUI
- GUI stands for Graphical user interface
- Example − Desktop, Mobile Devices, laptops

Fig. 1.11 Presentation Server

1.4.2.2 Application Servers

Application servers include specialized systems with multiple CPUs and a vast amount of RAM.

- Application Layer is also known as Kernel Layer and Basic Layer.
- SAP application programs are executed in Application Layer.
- Application Layer serves as a purpose of a communicator between Presentation and Database Layer.
- Application server is where the dispatcher distributes the work load to the different work processes makes the job done.

Fig. 1.12 Application Server

1.4.2.3 Database Servers

Database servers contain specialized systems with fast and large hard-drives.

- Database layer stores the data
- Data store can be Business data, SAP system data, SAP tables, Programs.
- Examples − Oracle, Microsoft SQL Server, IBM DB/2, Siebel, Sybase, etc.

Fig. 1.13 Database Server

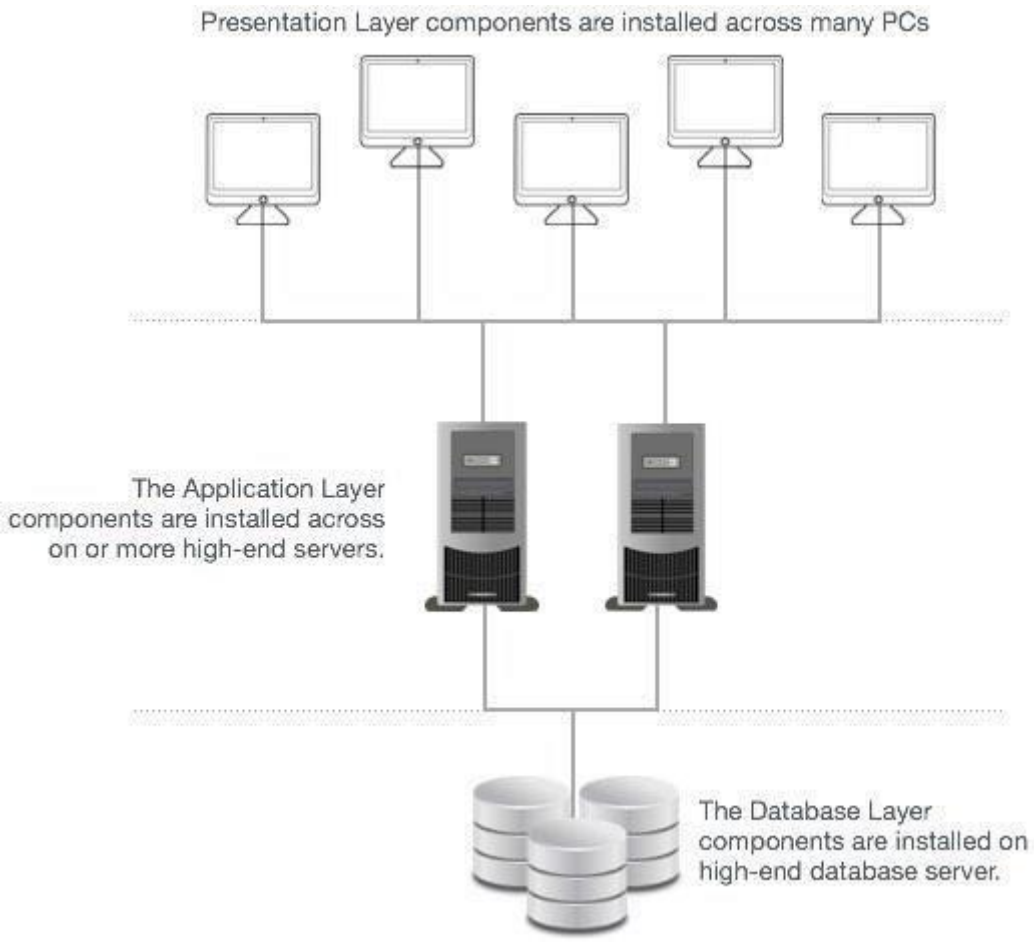


Fig. 1.14 Relation between Presentation, Application and Database Layers

1.5 NetWeaverata Glance

SAP NetWeaver describes all the software and services used for 'Business Enablement'. The SAP Business suite, such as ECC or SRM, contains the software components for that specific business solution.

- SAP NetWeaver is an open technology platform that offers a comprehensive set of technologies for running mission-critical business applications and integrating people, processes, and information.
- SAP NetWeaver is a web-based, open integration, application platform that serves as the foundation for enterprise service-oriented architecture (enterprise SOA) and allows the integration and alignment of people, information, and business processes across business and technology boundaries.
- It utilizes open standards to enable integration with information and applications from almost any source or technology.

- SAP NetWeaver is the foundation of SAP Business Suite and SAP Business by Design. It also powers partner solutions and customer custom-built applications.

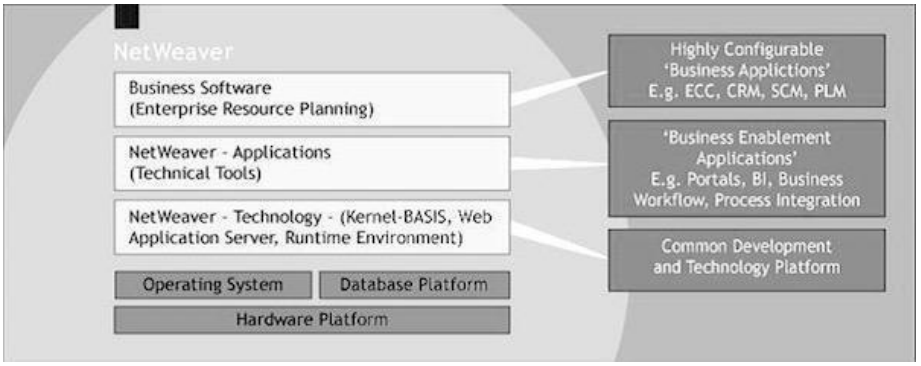


Fig. 1.15 SAP NetWeaver

1.5.1 SAP NetWeaverComponents

SAP NetWeaver includes a comprehensive set of components, applications, and tools.

1. **SAP NetWeaver Application Server**

It supports platform-independent web services, business applications, and standards-based development, enabling you to leverage existing technology assets for Web-services-oriented solutions.

2. **SAP NetWeaver Business Warehouse**

It enables you to integrate data from across the enterprise and transform it into practical, timely business information to drive sound decision making.

3. **SAP NetWeaver Gateway**

It enables developers to create applications that link business users to SAP software from any environment and through any device.

4. **SAP NetWeaver Master Data Management**

It ensures cross-system data consistency and helps integrate business processes across the extended value chain.

5. **SAP NetWeaver Process Orchestration**

It helps improve processes, from simple workflows to integrated processes that span applications and organizational boundaries. It includes capabilities for business process management, business rules management, and process integration.

6. **SAP NetWeaver Portal**

It unifies critical information and applications to give users role-based views that span the enterprise, enabling you to take full advantage of your information resources.

7. **SAP Auto-ID Infrastructure**

It gives you all the capabilities you need to integrate all automated sensing devices including RFID readers and printers, Bluetooth devices, embedded systems, and barcode devices.

8. **SAP NetWeaver Identity Management**

It addresses access and provisioning issues facing a typical enterprise. It creates a new opportunity for integrating business processes, and helps you to integrate systems in a heterogeneous IT environment.

9. **SAP NetWeaver Information Lifecycle Management**

It allows you to archive data in a readily accessible format according to regulatory retention rules that you define.

1.5.2 SAP NetWeaverTools

SAP NetWeaver includes the following tools −

1. **Adaptive Computing Controller**

It provides a central point of control for assigning computing resources and optimizing their use.

2. **SAP NetWeaver Composition Environment**

It provides a robust environment for design, deployment, and running of composite applications that comply with a service-oriented architecture.

3. **SAP NetWeaver Developer Studio**

It offers a convenient user interface and rich functionality for developing J2EE applications.

4. **SAP NetWeaver Visual Composer**

It simplifies the creation of portal content and analytics applications, enabling business analysts to build or customize applications using a visual user interface rather than manual coding.

5. **SAP Solution Manager**

It facilitates technical support for distributed systems with functionality that covers all key aspects of solution deployment, operation, and continuous improvement.

1.5.3 SAP NetWeaverApplications

SAP NetWeaver includes the following applications −

a) **SAP NetWeaver Enterprise Search**

It provides a simple and secure gateway to enterprise objects and transactions.

b) **SAP NetWeaver Single Sign-On**

It offers a comprehensive single sign-on solution, enabling reuse of a person's initial authentication for subsequent log-ins to all applications.

CHAPTER 2

INTRODUTION TO ABAP

ABAP (Advanced Business Application Programming), is a fourth-generation programming language, used for development and customization purposes in the SAP software.

2.1 Login Screen

After you log on to SAP server, SAP login screen will prompt for User ID and Password. You need to provide a valid user ID and Password and press Enter (the user id and password is provided by system administrator). Following is the login screen.

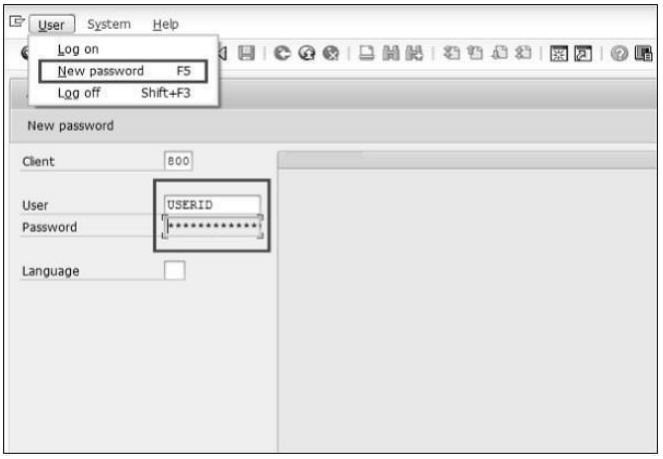


Fig. 2.1 Login Screen

2.2 Toolbar Icon

Following is the SAP screen toolbar.

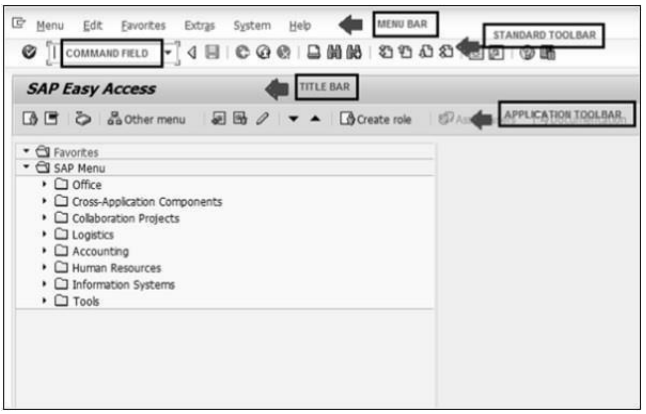


Fig. 2.2 Screen Toolbar

Menu Bar − Menu bar is the top line of dialog window.

Standard Toolbar − Most standard functions such as Top of Page, End of Page, Page Up, Page Down and Save are available in this toolbar.

Title Bar − Title Bar displays the name of the application/business process you are currently in.

Application Toolbar − Application specific menu options are available here.

Command Field − We can start an application without navigating through the menu transactions and some logical codes are assigned to business processes. Transaction codes are entered in the command field to directly start the application.

2.3 ABAP Editor

You may just start the transaction SE38 (enter SE38 in Command Field) to navigate to the ABAP Editor.

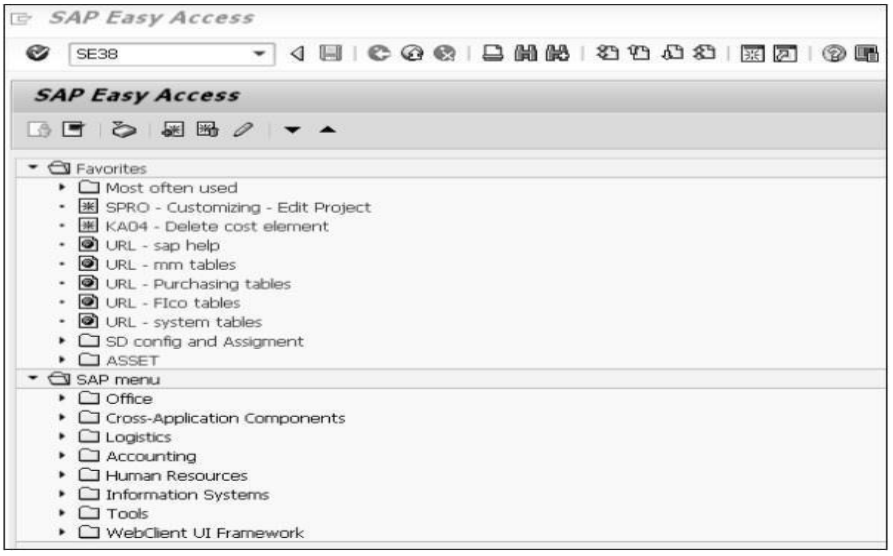
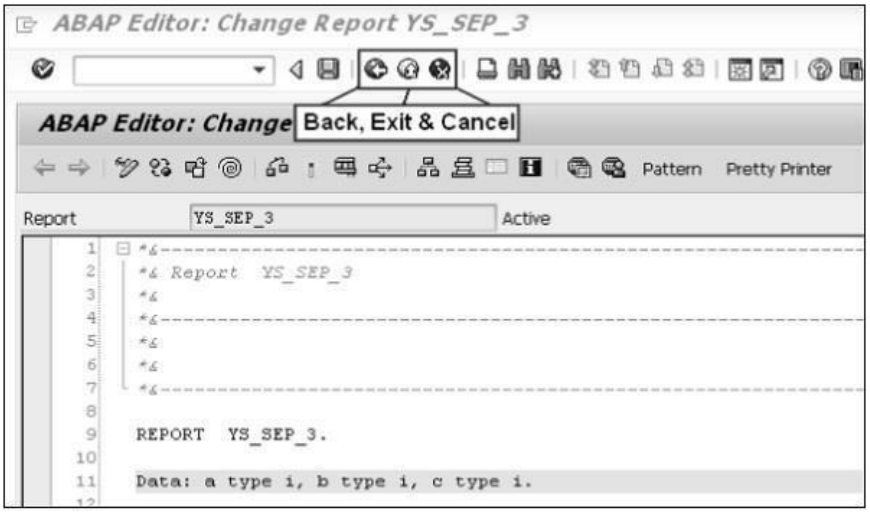


Fig. 2.3 SAP Easy Access Screen.

2.4 Standard Keys and Icons

Exit keys are used to exit the program/module or to log off. They are also used to go back to the last accessed screen.

Following are the standard exit keys used in SAP as shown in the image.



Following are the options for checking, activating and processing the reports.

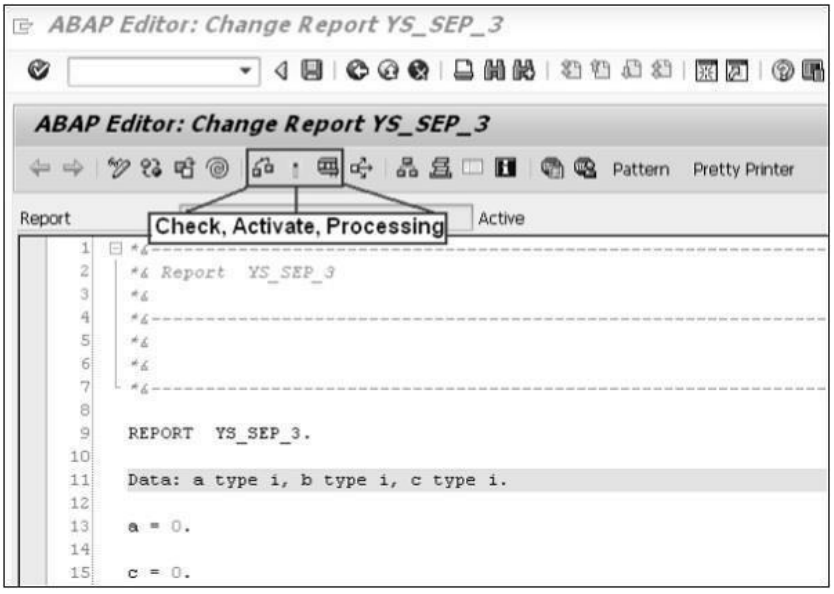


Fig. 2.4 Standard Keys and Icons

2.5 Data Types

ABAP offers the programmer a rich assortment of fixed length as well as variable length data types. Following table lists down ABAP elementary data types −

Type	Keyword
Byte field	X
Text field	C

Integer	I
Floating point	F
Packed number	P
Text string	STRING

2.6 OPERATORS

ABAP provides a rich set of operators to manipulate variables. All ABAP operators are classified into four categories −

- Arithmetic Operators
- Comparison Operators
- Bitwise Operators
- Character String Operators

2.7 ABAP Dictionary

The basic types in ABAP Dictionary are as follows −

- **Data elements** describe an elementary type by defining the data type, length and possibly decimal places.
- **Structures** with components that can have any type.
- **Table types** describe the structure of an internal table.

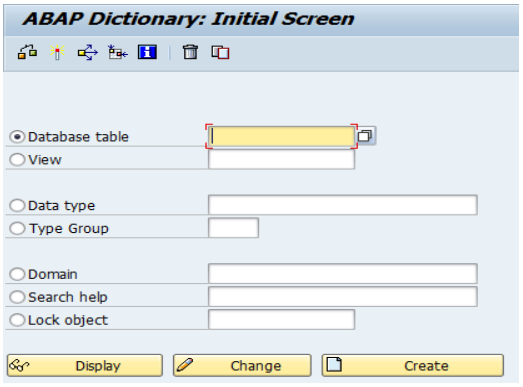


Fig. 2.5 ABAP Dictionary Initial Screen.

Various objects in the Dictionary environment can be referenced in ABAP programs. The Dictionary is known as the global area. The objects in the Dictionary are global to all ABAP programs and the data in ABAP programs can be declared by reference to these Dictionary global objects.

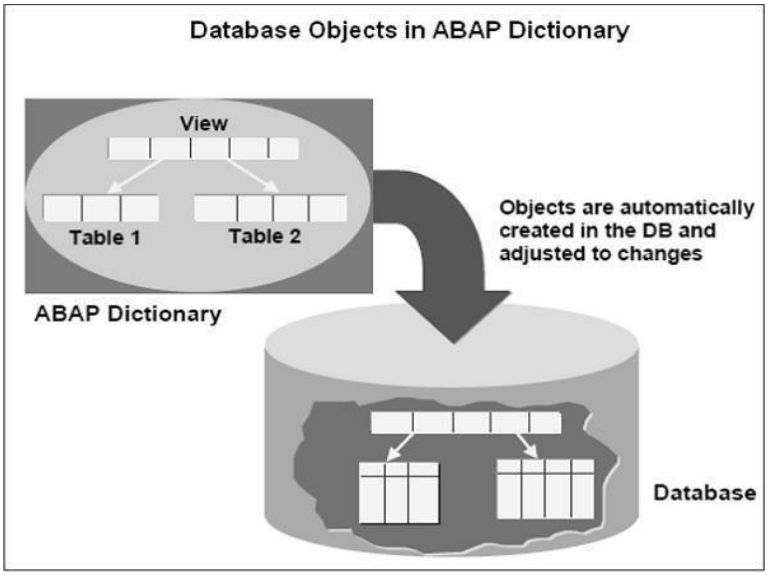


Fig. 2.6 Database objects in Dictionary

Fig. 2.7 Dictionary Objects

The Dictionary supports the definition of user-defined types and these types are used in ABAP programs. They also define the structure of database objects such as tables, views and indexes. These objects are created automatically in the underlying database in their Dictionary definitions when the objects are activated. The Dictionary also provides editing tools like Search Help and locking tool like Lock Objects.

2.8 MODULARIZING

ABAP programs are made up of processing blocks known as modularizing processing blocks. They are −

- The processing blocks called from outside the program and from the ABAP run-time environment (i.e., event blocks and dialog modules).
- Processing blocks called from ABAP programs.
  Apart from the modularization with processing blocks, source code modules are used to modularize your source code through macros and include programs.

1. **Modularization at source code level −**
   - Local Macros
   - Global Include programs
2. **Modularization through processing blocks called from ABAP programs −**
   - Subroutines
   - Function modules

Modularizing a source code means placing a sequence of ABAP statements in a module. The modularized source code can be called in a program as per the requirement of the user. Source code modules enhance the readability and understandability of ABAP programs. Creating individual source code modules also prevents one from having to repeatedly write the same statements again and again that in turn makes the code easier to understand for anyone going through it for the first time.

2.9 OBJECT ORIENTED CONCEPTS

ABAP was initially developed as a procedural language (just similar to earlier procedural programming language like COBOL). But ABAP has now adapted the principles of object oriented paradigms with the introduction of ABAP Objects. The object-oriented

concepts in ABAP such as class, object, inheritance, and polymorphism, are essentially the same as those of other modern object-oriented languages such as Java or C++.

As object orientation begins to take shape, each class assumes specific role assignments. This division of labor helps to simplify the overall programming model, allowing each class to specialize in solving a particular piece of the problem at hand. Such classes have high cohesion and the operations of each class are closely related in some intuitive way.

The key features of object orientation are −

- Effective programming structure.
- Real-world entities can be modeled very well.
- Stress on data security and access.
- Minimizes code redundancy.
- Data abstraction and encapsulation.

2.10 WEB DYNPRO

Web Dynpro (WD) for ABAP is the SAP standard user interface technology developed by SAP AG. It can be used in the development of web-based applications in the SAP ABAP environment that utilizes SAP development tools and concepts. It provides a front-end web user interface to connect directly to backend SAP R/3 systems to access data and functions for reporting.

Web Dynpro for ABAP consists of a run-time environment and a graphical development environment with specific development tools that are integrated in the ABAP Workbench (transaction: SE80).

2.11 TRANSACTION CODES IN SAP ABAP

- **SA38** Execute a program.
- **SCAT** Computer Aided Test Tool
- **SCU0** Compare Tables
- **SE01** Old Transport & Corrections screen
- **SE09** Workbench Organizer
- **SE10** Customizing Organizer
- **SE11** ABAP/4 Dictionary.

- **SE12** Dictionary: Initial Screen – enter object name
- **SE13** Access tables in ABAP/4 Dictionary.
- **SE14** ABAP/4 Dictionary: Database Utility.
- **SE14** Utilities for Dictionary Tables
- **SE15** ABAP/4 Repository Information System
- **SE16** Data Browser
- **SE16** Data Browser: Initial Screen.
- **SE16** Display table contents
- **SE17** General Table Display
- **SE30** ABAP/4 Runtime Analysis
- **SE30** ABAP/4 Runtime Analysis: Initial Screen.
- **SE30** Run Time Analysis (press Tips and Tricks button for good stuff)
- **SE32** ABAP/4 Text Element Maintenance
- **SE35** ABAP/4 Dialog Modules
- **SE36** ABAP/4: Logical Databases
- **SE37** ABAP/4 Function Library.
- **SE37** ABAP/4 Function Modules
- **SE38** ABAP Editor
- **SE38** ABAP/4 Program Development
- **SE39** Splitscreen Editor: Program Compare
- **SE41** Menu Painter
- **SE43** Maintain Area Menu
- **SE51** Screen Painter
- **SE51** Screen Painter: Initial Screen.
- **SE54** Generate View Maintenance Module
- **SE61** R/3 Documentation
- **SE62** Industry utilities
- **SE63** Translate Short/Long Text.
- **SE63** Translation
- **SE64** Terminology

CHAPTER 3

ABAP DICTONARY

3.1 INTRODUCTION

ABAP Dictionary can be viewed as metadata (i.e. data about data) that resides in the SAP database along with the metadata maintained by the database. The Dictionary is used to create and manage data definitions and to create Tables, Data Elements, Domains, View. You use the ABAP Dictionary to create and manage data definitions (metadata). The ABAP Dictionary permits a central description of all the data used in the system without redundancies. New or modified information is automatically provided for all the system components. This ensures data integrity, data consistency and data security.

The ABAP Dictionary supports the definition of user-defined types (data elements, structures and table types). You can create the corresponding objects (tables or views) in the underlying relational database using these data definitions. The ABAP Dictionary describes the logical structure of the objects used in application development and shows how they are mapped to the underlying relational database in tables or views

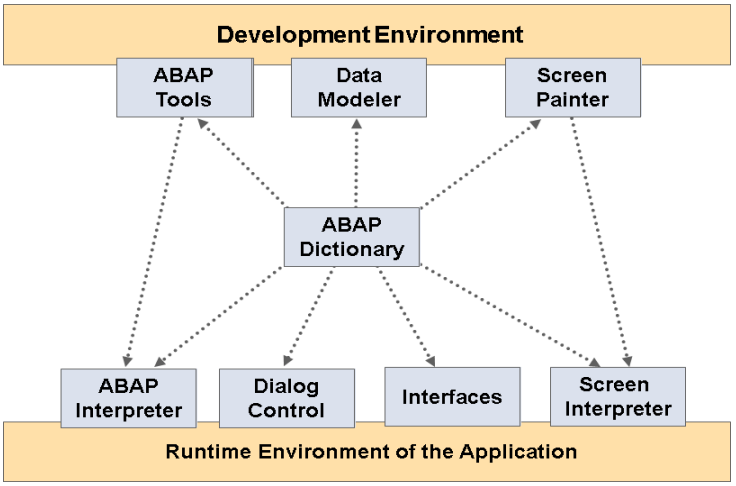


Fig. 3.1 Role of ABAP Dictionary

3.2 FUNCTIONS OF ABAP DICTIONARY

The ABAP Dictionary performs three key functions: Type Definitions – create user defined types, such as data elements, structures DB Objects – create tables, index and views Services that support program development – ex setting and releasing locks, defining input help and attaching a field help to a screen field.

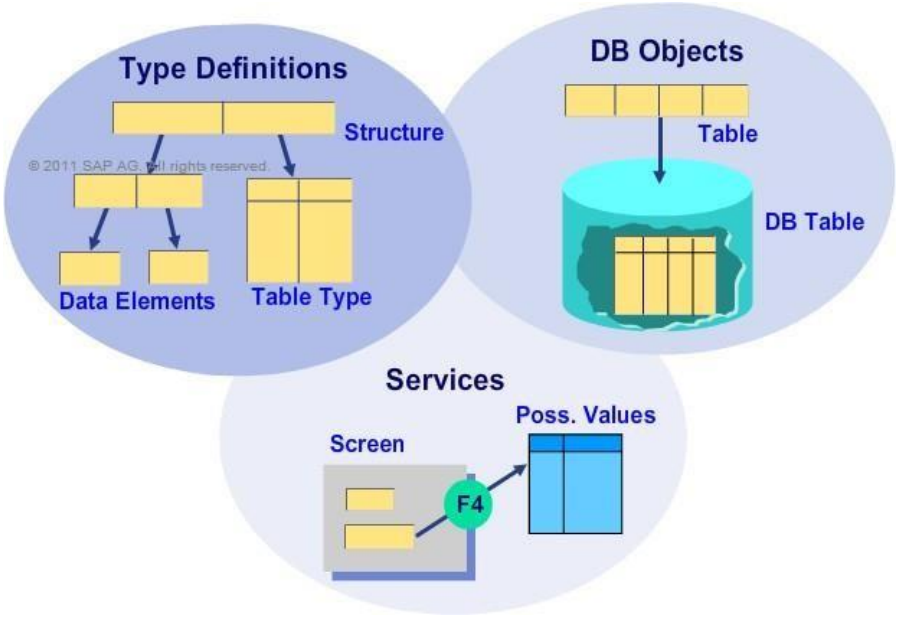


Fig. 3.2 Functions of ABAP Dictionary

The ABAP Dictionary enables central management of all the type definitions used in the R/3 System. You can use this tool to create data types, such as data elements, structures, and table types, either for ABAP programs or for applications, such as function modules and object methods. The ABAP Dictionary also defines database objects such as tables, indexes, and views. It also provides numerous services that support program development, such as setting and releasing locks, defining an input help, and attaching a field help to a screen field.

3.3 ABAP DATATYPES

A formal description of a variable is called a data type. When you define a variable or constant concretely by means of a data type, you define a data object. The type of a data object defines its technical (and possibly also semantic) properties. The type of an interface parameter is essential for determining the type of the actual parameter that is transferred when the modularization unit is called. An input or output field should also be

assigned a type. Only then can it provide information in addition to the technical characteristics, such as the field and value input h

3.3.1 ABAP STANDARD TYPES: COMPLETE

D
Type for date(D), format: YYYYMMDD, length 8 (fixed)

T
Type for time (T), format: HHMMSS, length 6 (fixed)

I
Type for integer (I), length 4 (fixed)

F
Type for floating point number (F), length 8 (fixed)

DECFLOAT16
Type for DECimal FLOATing point number, length 8 (fixed) - available as of AS ABAP 7.02

DECFLOAT34
Type for DECimal FLOATing point number, length 16 (fixed) - available as of AS ABAP 7.02

STRING
Type for dynamic length character string

XSTRING
Type for dynamic length byte sequence (HeXadecimal string)

The ABAP standard types predefined by SAP are divided into two groups: complete and incomplete types. Complete types are type-specific and have a fixed-length specification: Type **D** is used for date with format **YYYYMMDD** and fixed length **8**. **T** is used for time with format **HHMMSS** and fixed length **6**. **I** is used for integer and has a fixed length of **4**. **F** is used for floating point number and has a fixed length of **8**. **DECFLOAT16** or **DEC**imal **FLOAT**ing point number has a fixed length of **8** and is available as of AS ABAP 7.02. **DECFLOAT34** or **DEC**imal **FLOAT**ing point number has a fixed length of **16** and is available as of AS ABAP 7.02.**STRING** is the type used for dynamic length character strings. **XSTRING** or he**x**adecimal string is the type used for dynamic length byte sequence.

3.3.2 ABAP STANDARD TYPES: INCOMPLETE

C

Type for character string (Character) for which the fixed length is to be specified

N

Type for numerical character string (Numerical character) for which the fixed length is to be specified

X

Type for byte sequence (HeXadecimal string) for which the fixed length is to be specified

P

Type for packed number (Packed number) for which the fixed length is to be specified.

For some ABAP standard types, you have to specify a fixed length of the variable while defining a data object. These **incomplete** ABAP standard types do not contain a fixed length: **C** is used for the character string (**C**haracter). **N** is used for the numerical character string (**N**umerical character). **X** is used for byte sequence (He**X**adecimal string). **P** is used for packed number (**P**acked number). In the definition of a packed number, the number of decimal points may also be specified.

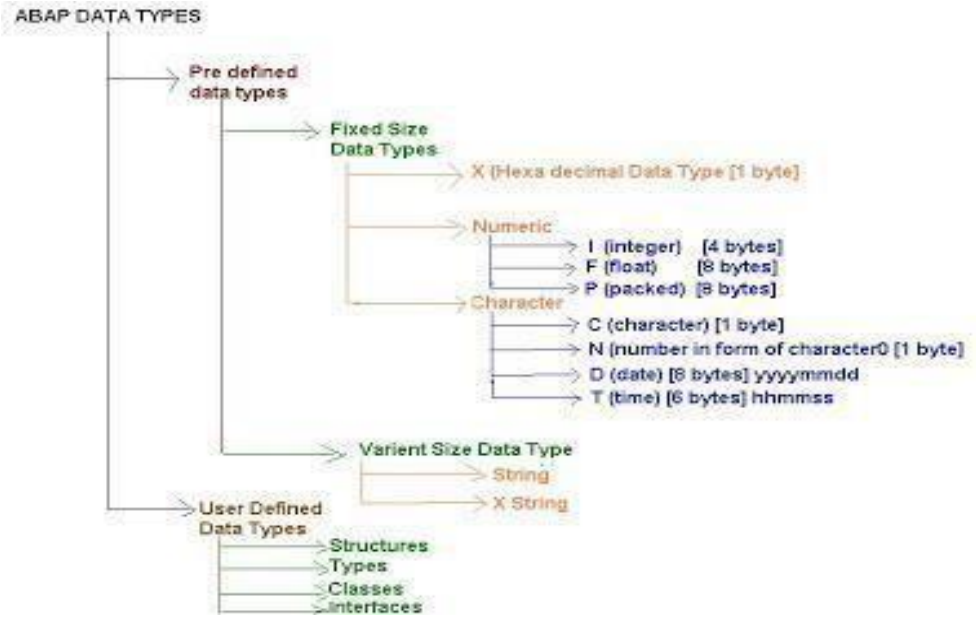


Fig. 3.3 ABAP Data Types

1. Integer Data Type occupies 4 bytes of memory and we can store up to 10 digits in an integer.

2. Float data type stores 8 bytes of memory. This stores decimal data also. It is defined with 16 decimal places by default.
3. Packed data type is same as Float data type except that by default, there are no decimal places and ABAPER has to specify the required no. of decimalplaces.
4. Character data type can accept all ASCII characters and it occupies 1 byte of memory.
5. Number accepts only numbers but treats them as characters. i.e., string operations can be executed
6. Date data type is of 8 bytes long and stores date in YYYYMMDD format. Time data type is of 6 bytes long and stores time in HHMMSS format.
7. String data type has variable size and it adjusts based on the data in the string. It can read only .txt files
8. Xstring is the hexadecimal formatted and similar to String type. i.e., it can read all files like .doc, .ppt, .xcs, .jpg, .bmp etc.
9. We can also create our own Data Types using "Types" or by creating custom objects like Structures, Classes, Interfaces etc.

3.3.3 Differences between Float and Packed data types:

a) FLOAT has default decimal places of 16 & PACKED data type has 0.
b) FLOAT decimal places are fixed whereas can be decided by the ABAP for PACKED.
c) PARAMETER, which creates GUI for data type is not supported by FLOAT while it is supported by PACKED type.

3.3.4 User defined data types

Use **TYPES** keyword to define the data types. TYPES: name(10) TYPE c,

length TYPE p DECIMALS 2, counter TYPE i, id(5) TYPE n.

3.3.5 Structured data types

Structured data type is grouping of several simple data types under one name. Use the keywords **BEGIN OF** and **END OF** to create a structured data type.

TYPES: BEGIN OF student, id(5)

TYPE n, name(10) TYPE c, dob TYPE d, place(10) TYPE c.

END OF student.

3.3.6 Constants

Constants are used to store a value under a name. We must specify the value when we declare a constant and the value cannot be changed later in the program. Use **CONSTANTS** keyword to declare a constant.

CONSTANTS: pi TYPE p DECIMALS 2 VALUE '3.14', yes TYPE c VALUE 'X'.

3.3.7 ABAP Variables

ABAP variables are i n s t a nc e s of data types. Variables are created during program execution and destroyed after program execution.

Use keyword **DATA** to declare a variable.

DATA: firstname(10) TYPE c, index TYPE i, student_id(5) TYPE n.

While declaring a variable we can also refer to an existing variable instead of data type.

3.4 GLOBAL TYPES IN THE DICTIONARY

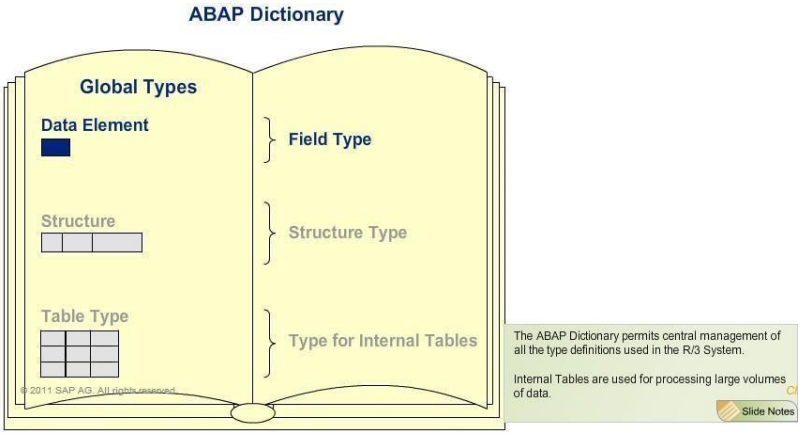


Fig. 3.4 ABAP Dictionary

A data type defined in the **ABAP Dictionary** is called global and can be used throughout the SAP system. Global Types include Data Element, Structure, and Table Type. Data Elements define Field Types. Structure and Table Type are used to define a structure type and a type for internal tables, respectively.

3.4.1 DEFINING DATA OBJECTS

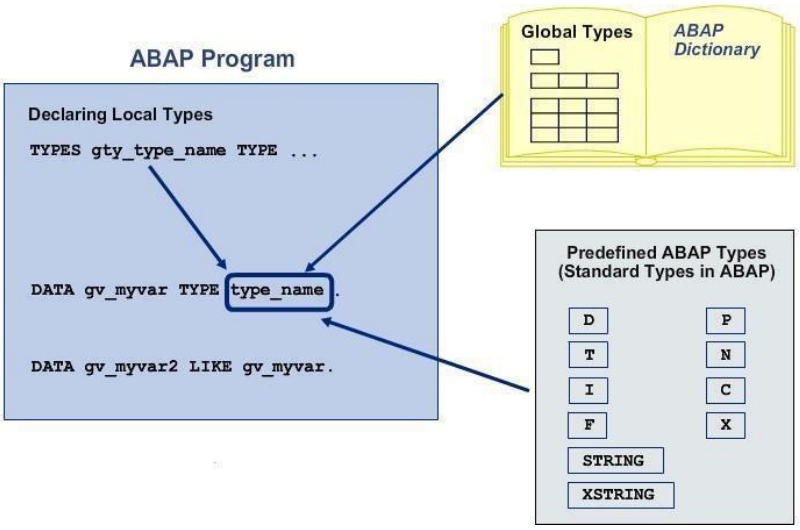


Fig. 3.5 Defining Dictionary Data Objects

Data objects are always defined with the DATA keyword. You can use an ABAP standard type, a local type, or a global type to define a type for the data object.

You can refer to an already defined data object when defining additional variables by using the keyword LIKE.

Example:

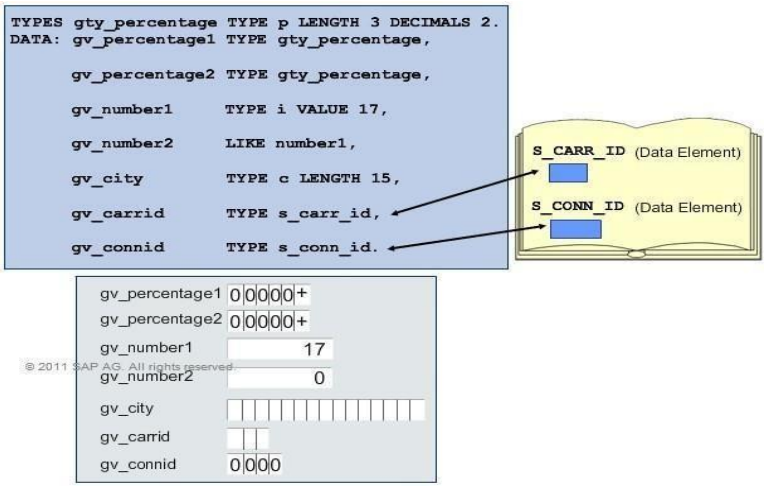


Fig. 3.6 Example

You can use the VALUE addition to pre assign the value of an elementary data object.

If the length is not specified in the variable definition, a default length for the incomplete standard type is used. The default length is 1 for types C, N, and X, and 8 for type P.

A data object without a type and length specification takes a default type C with length 1.

3.4.2 DATA OBJECTS IN ABAP DICTIONARY

DATA DICTIONARY INITIAL SCREEN (SE11)

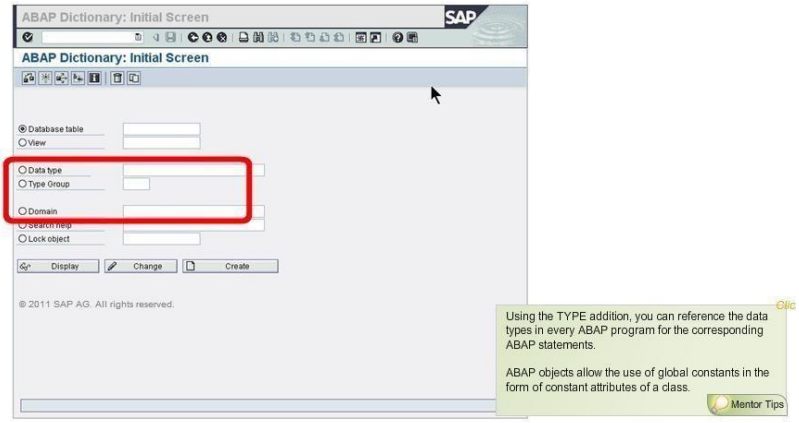


Fig. 3.7 Data Dictionary Initial Screen

The globally known data types are defined using the Data Type field.

You can define the type group for creating global constants using the Type Group field. The Domain field is where the technical properties of data objects can be defined.

3.4.3 DATA TYPES IN THE ABAP DICTIONARY

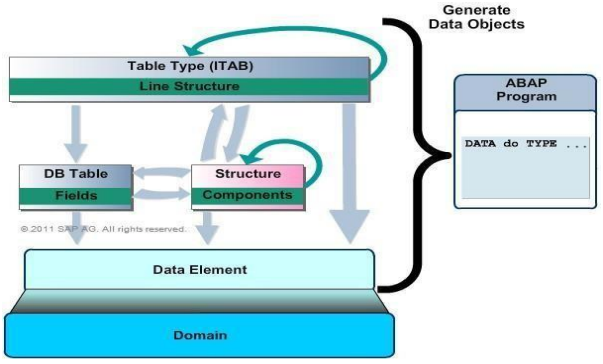


Fig. 3.8 Data types in ABAP Dictionary

You can generate data objects in the ABAP programs using the data types declared in the ABAP dictionary. However, an ABAP program cannot access domains for defining data objects. The data elements get their data types from domains.

The components of a structure can be elementary fields, tables, as well as structures.

A structured type or an elementary type can be the line type of a table type (ITAB).

3.4.3.1 DOMAIN

Domains cannot be used directly in programs or tables. Therefore, data element is used as a bridge from the technical properties to the different data types or data objects. You can define technical attributes and value ranges.

The most frequently used data types are CHAR, DATS, DEC, and NUMC. CHAR, the character string, can have a maximum length of 255 in tables. DATS defines the date and the length is set at eight characters.

DEC is used for the calculation of an amount in a field, and contains operators, such as the point, the plus or minus (+/-) sign, and thousand-separator commas. It can have a maximum of 31 characters.

NUMC is used for character strings that comprise only numbers. It is restricted to a maximum of 255 characters.

3.4.3.2 DATA ELEMENT

Data elements not only constitute the link between domains and the data objects, but also contain semantic or technical information about the data objects created from these.

You can translate the field labels into other languages by choosing the transaction SE63 or the menu path: **Goto → Translation**.

You can append F4 or input help to a data element.

You can assign the SET or GET parameter to the data element, if an input field based on the same data element exists on a subsequent screen. This helps you read the value from the parameter and enter it in the screen field.

Domains manage technical properties of data elements centrally. Data elements define the data types which can be used on screen or program etc.

3.4.3.3 STRUCTURES

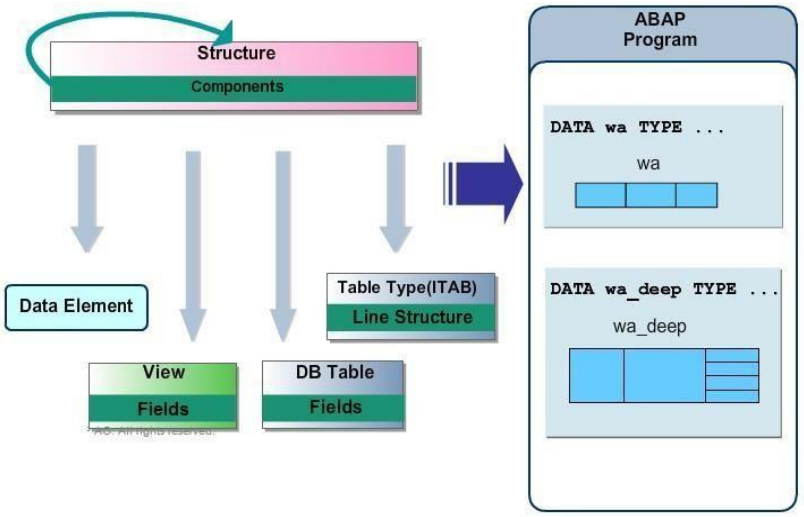


Fig. 3.9 Structures in ABAP Dictionary

Structure components can be data elements, integrated types, structure definitions of internal tables, DB table views, or other existing structure definitions. The data object, generated by including the fields of an actual two-dimensional object like a view or DB tables, remains flat or one-dimensional.

3.4.3.4 INTERNAL TABLES

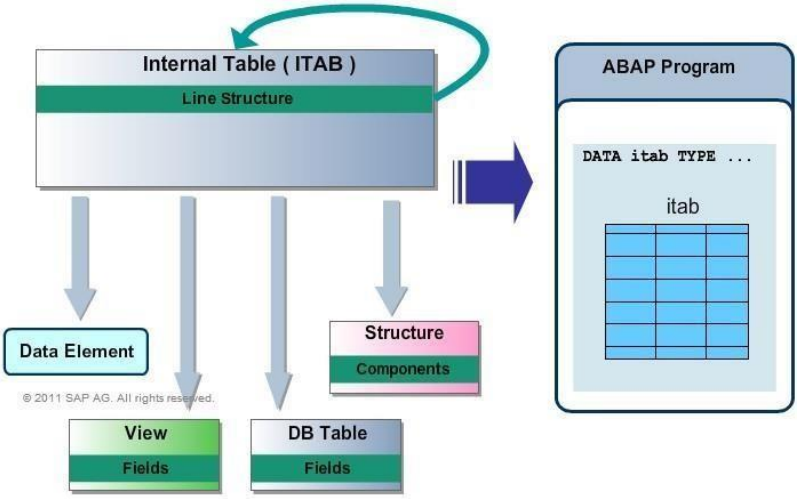


Fig. 3.10 Internal Table in ABAP Dictionary

Internal tables can be defined using an existing line structure. You can use database tables, structure definitions, views, data elements, direct type definitions, or existing table types as line types.

A two-dimensional array is created in the main memory by declaring an internal table in an ABAP program.

3.4.3.5 TYPE GROUP

type group used to define global, complex data types and global constants

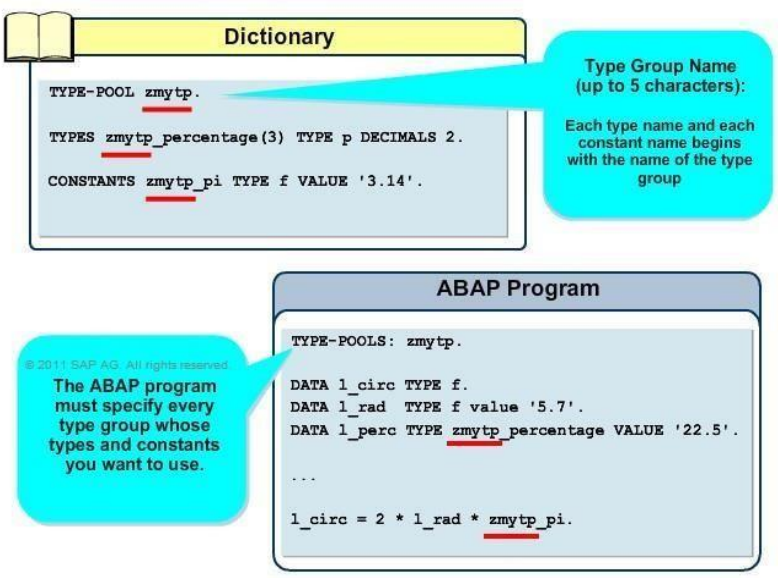


Fig. 3.11 Type Group in ABAP Dictionary

You declare the type group using the TYPE POOL statement. This enables you to use the types of a type group in a program.

All names of these data types and constants must begin with the name of the type group and an underscore.

CHAPTER 4

ABAP VIEWs

4.1 WHY DO YOU NEED VIEWS?

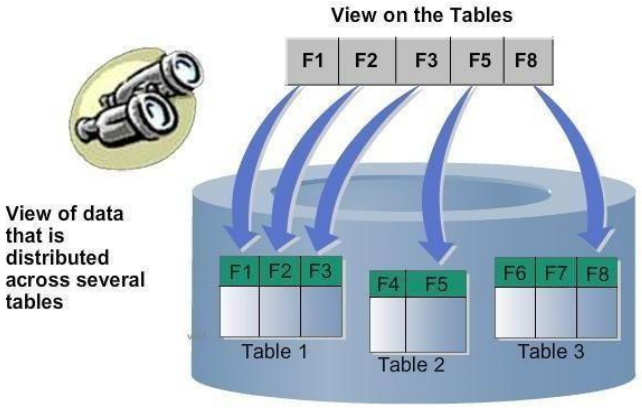


Fig. 4.1 ABAP Views

You typically find data for an application object distributed on several database tables. Therefore, you need database systems to define application-specific views on data in several tables. These are known as views. You can combine data from several tables in a meaningful way with the help of a view or join. Further, you can hide irrelevant information through projection. Alternatively, you can implement selection to display only those data records that satisfy certain conditions. You can display the data in a view in a format that is similar to the one used in the extended table maintenance.

4.2 STRUCTURE OF A VIEW - STARTING SITUATION

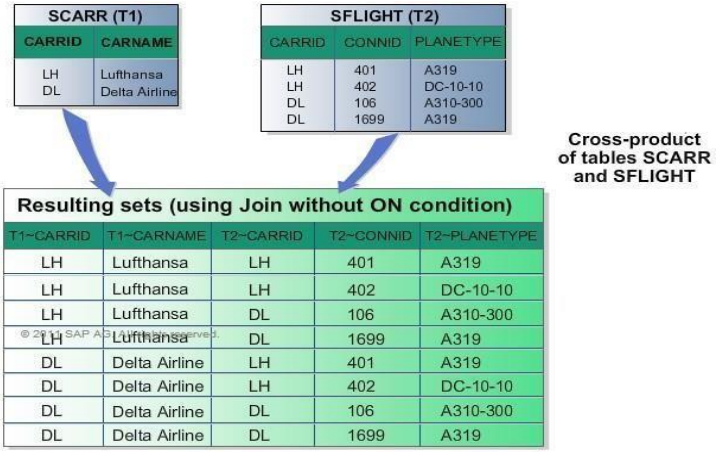


Fig. 4.2 Structure of ABAP Views

Let us understand the structure of a view and how to select data using this view. As shown here, table

SCARR contains two entries while there are four entries in table SFLIGHT.

These tables are first appended to one another. This creates a cross product of the two tables, in which each record in SCARR is combined with each record in SFLIGHT.

4.2.1 STRUCTURE OF A VIEW - JOIN CONDITION

Fig. 4.3 Structure of View – JOIN condition

The result in the cross product of the two tables needs to be limited with a join condition. This condition defines the relationship between the records of the two tables. Here, CARRID field for SCARR is compared with the CARRID field for SFLIGHT. The join condition is then: SCARR-CARRID = SFLIGHT-CARRID.

Hence, all the records whose entry in Field 1 is not identical to the entry in Field 3 are removed from the cross product. Therefore, the column for Field 3 is not required in the view.

4.2.1.1 STRUCTURE OF A VIEW - FIELD SELECTION (PROJECTION)

Fig. 4.4 Structure of View – PROJECTION conditionSometimes certain fields of a table involved in a view may not be relevant.

You can specifically define the set of fields to be included in the view or projection. As shown here, the PLANETYPE field is not relevant and can, therefore, be hidden.

4.2.2 STRUCTURE OF A VIEW - SELECTION CONDITION

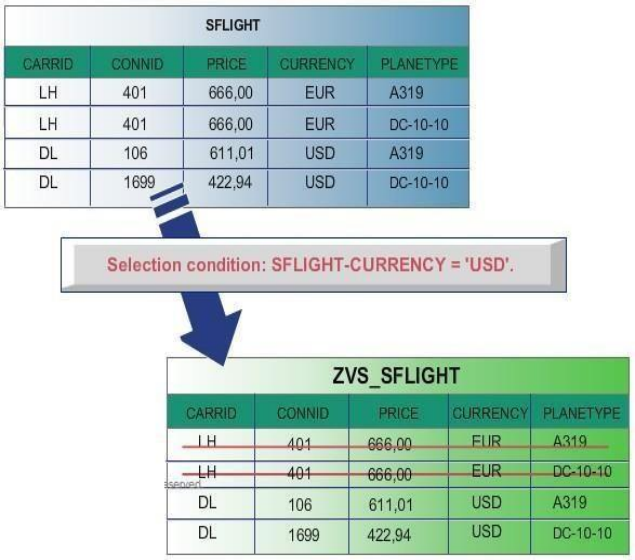


Fig. 4.5 Structure of View – SELECTION condition

You can use a selection condition to restrict the set of records that can be displayed with the view. As shown here, only those records with value DL in the CARRID field are required with the view. Therefore, a selection condition can also be formulated with a field that is not contained in the view.

4.3 DATABASE VIEWS

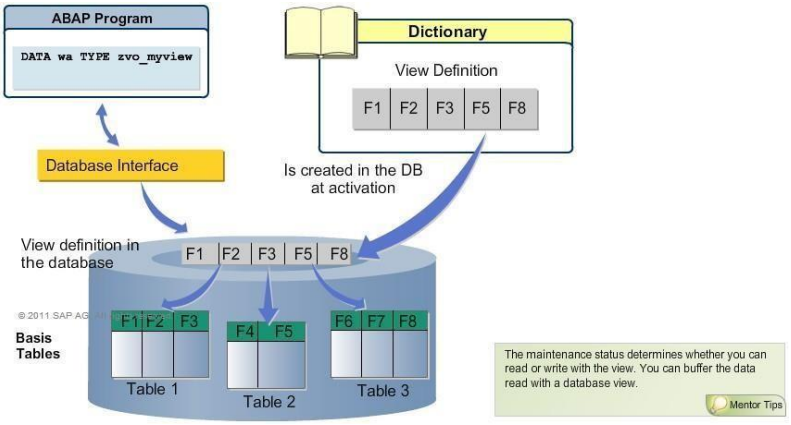


Fig. 4.6 Database View

A database view is created on the database during activation after it is defined in the ABAP Dictionary. Database interface directly passes access to the database view. If the

definition of a database view is changed in the ABAP Dictionary, the corresponding view created on the database must also be adjusted accordingly. The adjustment is made by deleting the old view definition and re-creating another on the database with a new definition. This is possible because a view does not contain any data.

4.3.1 DYNAMIC ATTACHMENT OF TABLE FIELDS IN DATABASE VIEWS

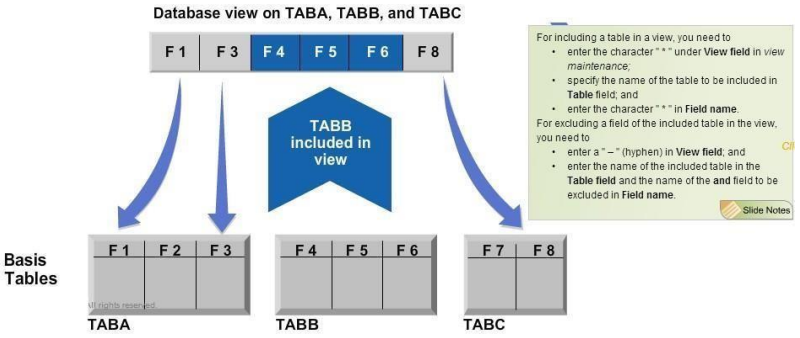


Fig. 4.7 Dynamic attachment of table fields in database views

If you include an entire table in a database view, then all the fields of the included table become the fields of the view. This allows you to explicitly exclude certain fields. Any changes made in the table are automatically reflected in the view. Therefore, if an append structure is added to a table included in a view, the fields added with the append structure also appear automatically in the view.

To include all fields in view, use * as field name. If you do not want to insert a field of the table in the view, you must enter a – in the view field.

Using an append view is similar to enhancing the table with an append structure. An append view is assigned to one database view. More than one append view can be created for one database view.

4.4 MAINTENANCE VIEW FROM A MAINTENANCE VIEW

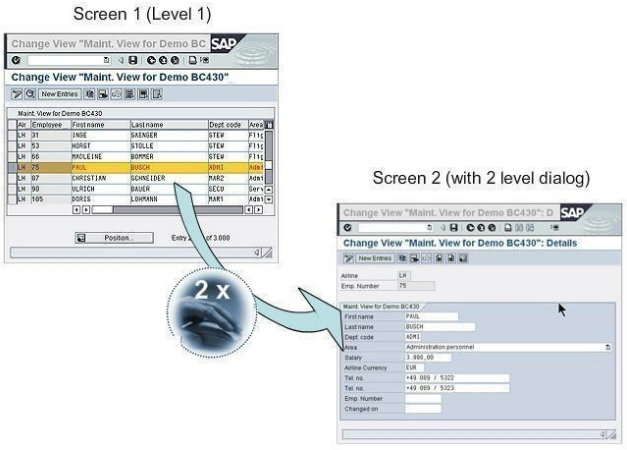


Fig. 4.8 Maintenance View

The first screen shows a one-step maintenance view which depicts the data of the table lines as Table Control. In this view, while you can change the function fields with a white background, you cannot do so for the fields having a gray background.

The second screen displays a two-step maintenance view which opens by double-clicking a table line in which the selected data record is presented clearly.

4.4.1 DEFINE MAINTENANCE VIEW

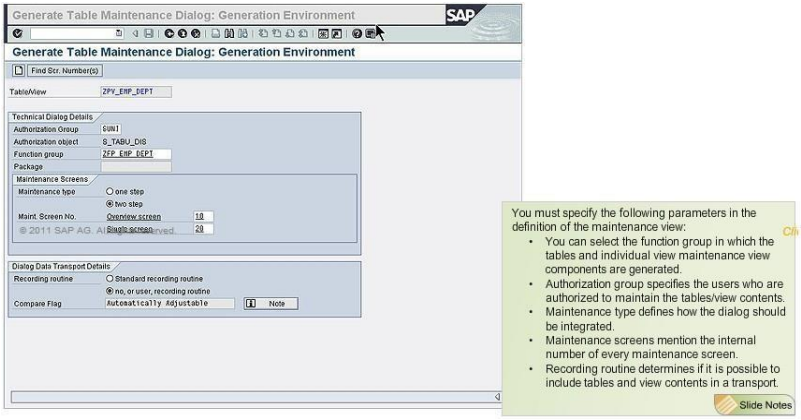


Fig. 4.9 Defining Maintenance View

You can arrive at the generation environment for maintenance views in the SE11 screen in the view for Tables/Views. To reach the initial screen of the maintenance transaction, select Utilities, then Table maintenance generator, then choose Development, followed by other tools, and finally select Table maintenance view.

Here, enter the name of the table or the view. Next, select generated objects and choose Create/Change. Subsequently, a prompt appears where you need to confirm that the maintenance modules should be created.

This commences the process of generating the maintenance view. For this, you first need to start the transaction SM30 and enter the table or view in the Table or View field for which you have generated the maintenance view. Lastly, activate the Maintenance button.

4.5 VIEW CLUSTERS AND MAINTENANCE VIEW

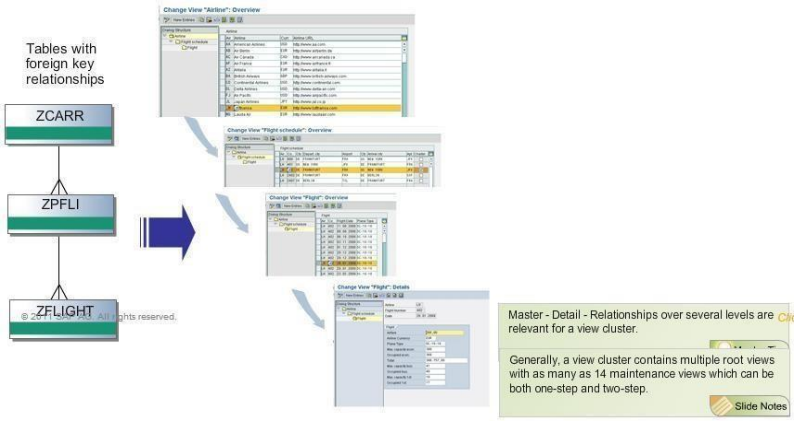
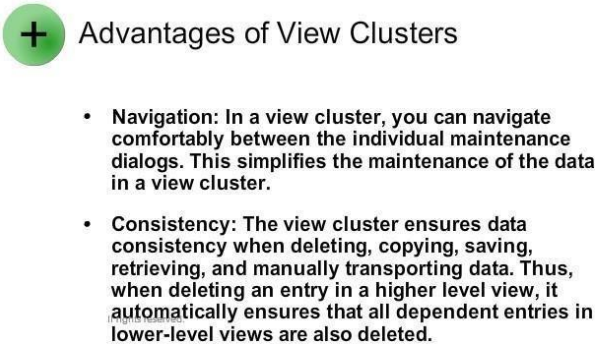


Fig. 4.10 View Clusters and Maintenance View

View clusters are a collection of maintenance views, which have been combined in one maintenance unit for either business or technical reasons. This enables the maintenance of data that are similar from the content point of view. You can navigate within the view cluster using the hierarchy of the tables or views underlying the individual views. It is necessary to generate a maintenance view for every table or view involved, which can in turn be combined in SE54 in a view cluster. The associated data can be maintained by specifying the cluster name with transaction SM34.

4.5.1 ADVANTAGES



a) One major advantage of view clusters is the ease of navigation. By using view clusters, you can easily navigate to each component of the object. This improves the maintenance of the data of a view cluster. Another advantage of view clusters is the consistency attained while deleting, copying, saving, retrieving, or manually transporting data. In fact, when an entry is deleted in a superior view, this is reflected in all the dependent entries in the subordinate view.

b) Maximum 14 maintenance dialogs can be combined into 1 view cluster.

c) While maintenance views can be used to maintain only database tables that have 1:1 relationship with one another, view clusters can be used for non-relationships

CHAPTER 5

ABAP SCREEN PROGRAMMING

5.1 INTRODUCTION TO ABAP SCREEN PROGRAMMING

Screens and their flow logic, which together form dynamic programs, are instances that control the flow of an ABAP program by calling a series of dialog modules. Screens can be combined into sequences, where the next screen in the sequence can be defined either statically or dynamically. The simplest screen sequence is a single screen. The sequence starts when you call its first screen. You can do this in a variety of ways.

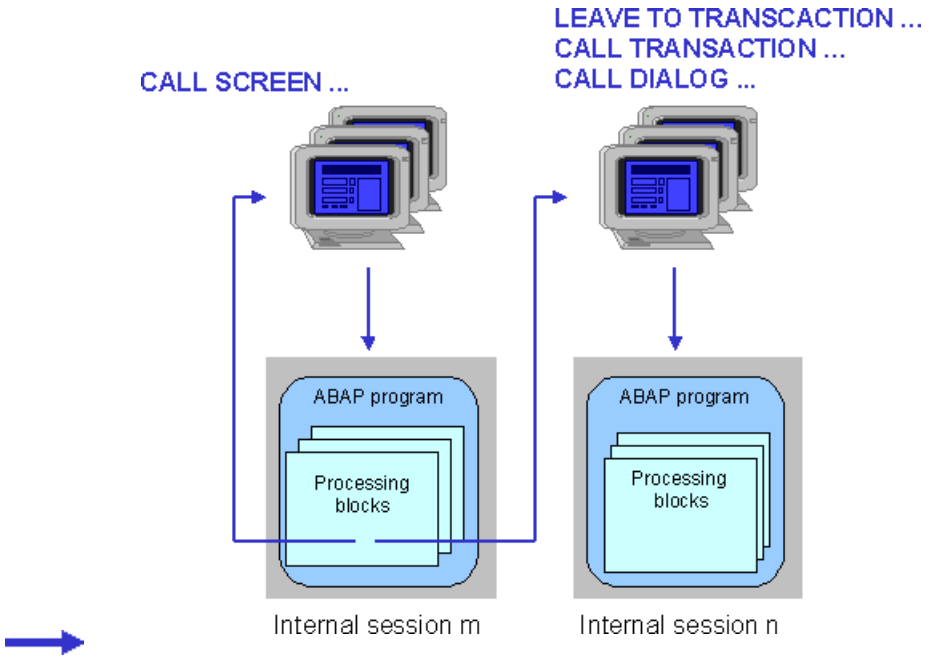


Fig. 5.1 Sequence of calling screens in Screen Programming

Simple ABAP dialog program creation- step by step process

Steps

1. Create dialog program using se38 transaction and giving type as module pool e.g ztest_prog
2. Create screen after clicking on display objects and then by right clicking on the module pool program e.g screen 1000. Remember to do save after every step

3. Create include files ztest_prog_o01(PBO process before include ztest_prog_o01) and ztest_prog_a01(PAI process after include file)
4. Uncomment the commented part in screen 1000 and double click on STATUS_1000(to create PBO module remember to associate include file ztest_prog_o01 file). similarly double click on USER_COMMAND_1000 (to create PAI module remember to associate include file ztest_prog_a01 file).
5. Uncomment the commented part in STATUS_1000(PBO module)and give some name to status bar and title bar
   SET PF-STATUS 'TESTSTATUS'.
   SET TITLEBAR 'TESTTITLE'.
   Double click on TESTSTATUS and TESTTITLE to create status bar and title bar
6. Write the code for PBO and PAI module. Sample code for PBO and PAI module is below.
7. Create the layout for screen

   Enter text boxes(for display purpose only can give any name) . Then enter input/output boxes(use the name of the box the same as used in the User_COMMAND_1000(PAI module)). Then enter push buttons give name like change, create and display but remember to give FCTCODE as given in the PBO module STATUS_1000.
8. Create transaction using the same name as used in PBO module give program name(ztest_prog main module program) and screen number (1000)while creating the transaction and check all three boxes for GUI support (HTML, java and windows). Create all three transactions
9. Finally activate everything by right clicking on the module and then executing the program by writing the transaction name in the command box for example(/oztest_create2)

5.2 SAMPLE PROGRAM

```
1 ************************************************************************
2 ********************
3 *PAI Module program
```

```
4  ********************
5  *&---------------------------------------------------------------------*
6  *&  Include          ZTEST_VIS_MP3_A01
7  *&---------------------------------------------------------------------*
8  *&---------------------------------------------------------------------*
9  *&      Module USER_COMMAND_1000 INPUT
10 *&---------------------------------------------------------------------*
11 MODULE user_command_1000 INPUT.
12 TABLES: z26empl.
13 DATA wa_po TYPE z26empl.
14 CASE sy-ucomm.
15 WHEN 'V_CREATE'.
16  wa_po-empid = z26empl-empid.
17  wa_po-grade = z26empl-grade.
18  wa_po-location = z26empl-location.
19  wa_po-pu = z26empl-pu.
20  MODIFY z26empl FROM wa_po.
21 MESSAGE 'Saved' TYPE 'I'.
22 WHEN 'V_CHANGE'.
23 wa_po-empid = z26empl-empid.
24 wa_po-grade = z26empl-grade.
25 wa_po-location = z26empl-location.
26 wa_po-pu = z26empl-pu.
27 MODIFY z26empl FROM wa_po.
28 WHEN 'V_DISPLAY'.
29 SELECT empid grade location pu
30 FROM z26empl
31 INTO CORRESPONDING FIELDS OF wa_po
32 WHERE empid = z26empl-empid.
33 z26empl-empid = wa_po-empid.
34 z26empl-grade = wa_po-grade.
35 z26empl-location = wa_po-location.
36 z26empl-pu = wa_po-pu.
```

```
CALL SCREEN '1000'.
ENDSELECT.
WHEN 'EXIT'.
LEAVE PROGRAM.
ENDCASE.
ENDMODULE.                 " USER_COMMAND_1000 INPUT
*&---------------------------------------------------------------------*
*&---------------------*
*pbo module program
*&---------------------*
MODULE status_1000 OUTPUT.
SET PF-STATUS 'VISHALSTATUS1'.
SET TITLEBAR 'VISHALTITLE1'.
LOOP AT SCREEN.
CASE screen-name.
WHEN 'CHANGE'.
IF sy-tcode NE 'ZTEST_CHANGE2'.
screen-invisible = 1.
MODIFY SCREEN.
ELSE.
screen-invisible = 0.
MODIFY SCREEN.
ENDIF.
WHEN 'CREATE'.
IF sy-tcode NE 'ZTEST_CREATE2'.
 screen-invisible = 1.
MODIFY  SCREEN.
ELSE.

```

```
70 screen-invisible = 0.
71 MODIFY SCREEN.
72
73 ENDIF.
74 WHEN 'DISPLAY'.
75 IF sy-tcode NE 'ZTEST_DISPLAY2'.
76 screen-invisible = 1.
77 MODIFY SCREEN.
78 ELSE.
   screen-invisible = 0.
   MODIFY SCREEN.
   ENDIF.
   ENDCASE.
   ENDLOOP.
   ENDMODULE.              " STATUS_1000  OUTPUT
```

5.3 Subscreens

A subscreen is an independent screen that is displayed in an area of another ("main") screen.

Subscreens allow you to embed one screen within another at runtime. You can include multiple sub- screens on main screen.

The term subscreen applies both to the screen that you embed, and the area on the main screen in which you place it. This tutorial is about subscreen areas. The actual screens created through SE51 transaction, are called subscreen screens if defined in screen attributes.

When you use a subscreen, the flow logic of the embedded screen is also embedded in the flow logic of the main screen. Hence, Using subscreens on screens is like using includes in ABAP programs.

To use a subscreen, you must follow three simple steps

a) Define the subscreen area(s) on a screen
b) Define suitable subscreen screens
c) Include the subscreen screen in the subscreen area.

Also, you need to adjust the frame of sub-screen and main screen. You need to name it in the field name field.

Further, you also need to adjust the fields within the subscreen to make them appear in main screen. In case the sub-screen is defined to be larger than the available area in the main screen, only the part of subscreen will be visible that fits in the area available. The area is always measured from the top left corner of screen. Hence you should take adequate care while defining sub-screen areas and creating sub-screens.

EXAMPLE

For instance here we have defined two sub-screen areas on main screen and have attached two different Sub-screen to corresponding areas. Whenever main screen is called, the PBO of main screen is called. But before display, the PBO's of each screen attached with sub-screen areas on main screen are also called.

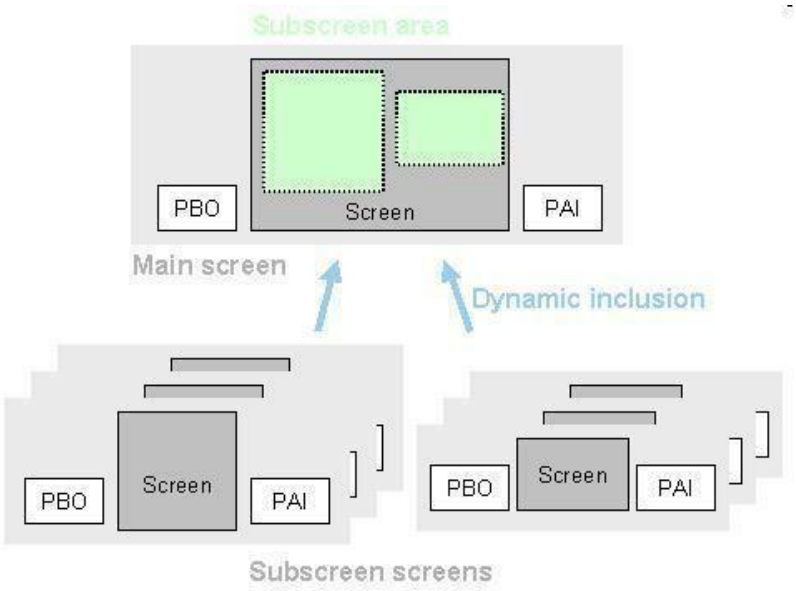


Fig. 5.2 Example of Screen Programming

You can include a subscreen screen using the CALL SUBSCREEN statement in the flow logic of the main screen.

To include a subscreen screen in the subscreen area of the main screen and call its PBO flow logic, use the following statement in the PBO event of the main screen:

5.3.1 PROCESS BEFORE OUTPUT.

CALL SUBSCREEN <area> INCLUDING [<prog>] <dynp>.

This statement assigns the subscreen screen with number <dynp> to the subscreen area called <area>. You can also specify the program in which the subscreen screen is defined (optional). If you do not specify the program explicitly, the system looks for the subscreen screen in the same ABAP program as the main program. If it does not find a corresponding subscreen screen, a runtime error occurs.

The PBO flow logic of the subscreen screen is also included at the same point. This can call PBO modules of the ABAP program in which the subscreen screen is defined. At the end of the subscreen PBO, the global fields from the program are passed to any identically-named screen fields in the subscreen screen. The PBO flow logic of the subscreen screen can itself include further subscreens.

The name <area> of the subscreen area must be entered directly without inverted commas. You can specify the names <prog> and <dynp> either as literals or variables. If you use variables, you must declare and fill identically-named variables in the ABAP program. The screen number <dynp> must be 4 characters long. If you do not assign a subscreen screen to an area, it remains empty.

To call the PAI flow logic of the subscreen screen, use the following statement in the PAI flow logic of the main screen:

5.3.2 PROCESS AFTER INPUT.

CALL SUBSCREEN <area>.

This statement includes the PAI flow logic of the subscreen screen included in the subscreen area <area> in the PBO event. This can call PAI modules of the ABAP program in which the subscreen screen is defined. Data is transported between identically-named fields in the subscreen screen and the ABAP program either when the PAI event is triggered, or at the corresponding FIELD statements in the PAI flow logic of the subscreen screen.

5.3.3 SUBSCREEN

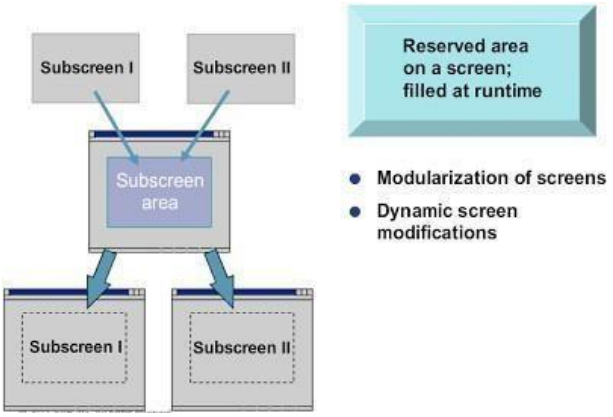


Fig. 5.3 Subscreens

A subscreen area is a reserved rectangular area on a screen, in which you add another screen during run time.

A second screen with the type subscreen must be created and then displayed in the subscreen area defined on the main screen. In other words, a subscreen is an independent screen displayed within another screen.

You can include multiple subscreens within a main screen. From a usability perspective, subscreens are highly attractive as different programs can use the same subscreens. You can also define the subscreens dynamically at run time.

5.3.3.1 SUBSCREEN AREA: ATTRIBUTE

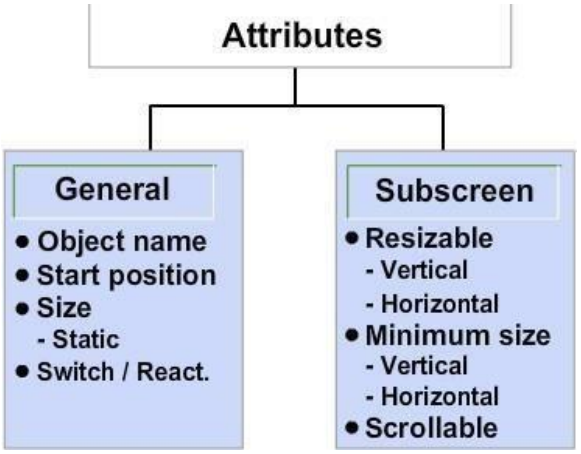


Fig. 5.4 Subscreen Area: Attribute

The general attributes of the subscreen area include its name, position, and size.

You can use the resizing attributes to determine whether the size of a subscreen area can be changed vertically and horizontally. If a subscreen is bigger than the subscreen area in which it is called, then only that part of the subscreen which can fit in the subscreen area is shown.

You can also use the minimum size attribute to set the lowest possible limit to which the subscreen area can be resized.

Similarly, by using the scrollable attribute, the system displays scrollbars for large screen sizes.

5.3.3.2 CREATING A SUBSCREEN AREA

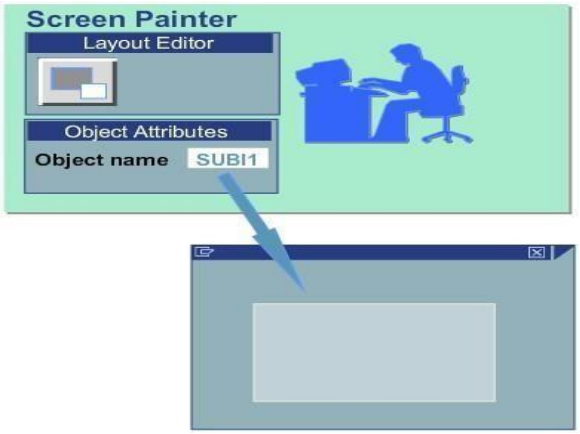


Fig. 5.5 Subscreen Area

For creating a subscreen area, you need to

- select a subscreen from the object list in the Screen Painter.
- enter a name for the subscreen area in the Object text field and position it on the screen.
- adjust the top-left corner of the table control area; and modify the size of theobject.

5.3.3.3 CALLING A SUBSCREEN

It is essential to call a subscreen in both the PBO and PAI sections of the flow logic of the main screen.

For this purpose, the ABAP modules for subscreens are programmed similarly as normal screens.

The CALL SUBSCREEN <subarea> statement executes the PBO and PAI processing blocks for the Subscreen as components of the PBO and PAI of the main screen.

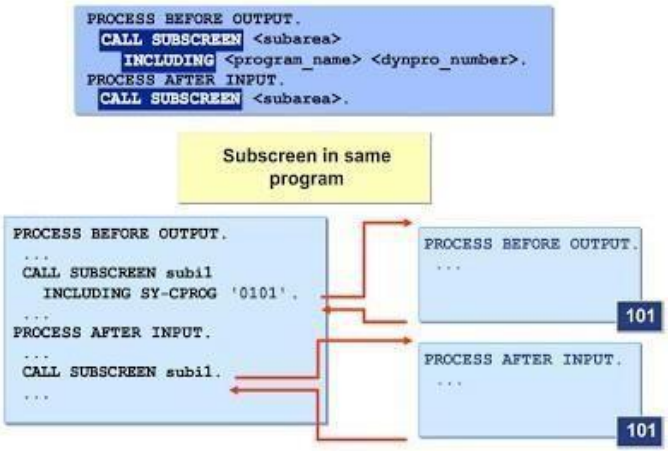


Fig. 5.6 Calling a subscreen

5.3.4 SUBSCREEN FROM EXTERNAL PROGRAM

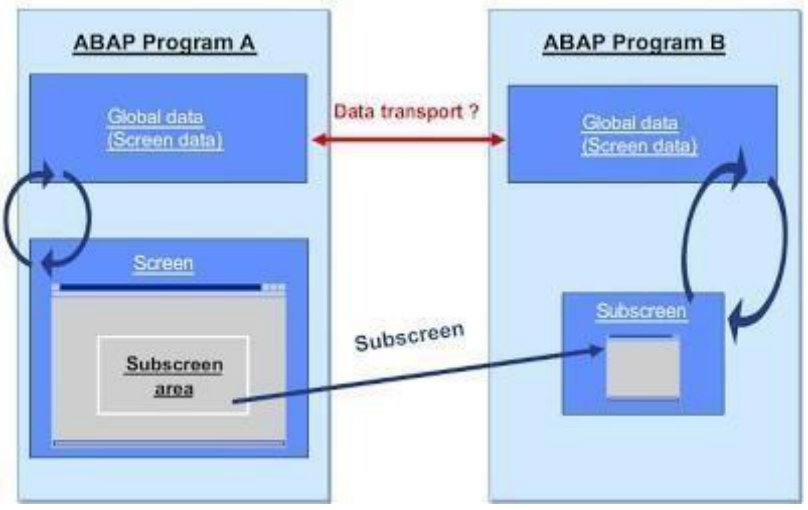


Fig. 5.7 calling subscreen from external program

You cannot access the global data of the main program in the subscreen if it is not in the same module pool as the main program. In this case, data transfer from the screen to the program does not occur.

Therefore, you must implement data transfer through a function module that exports and imports data with an appropriate MOVE statement in the subscreen coding.

5.3.5 SUBSCREENS: ENCAPSULATION IN FUNCTION GROUP

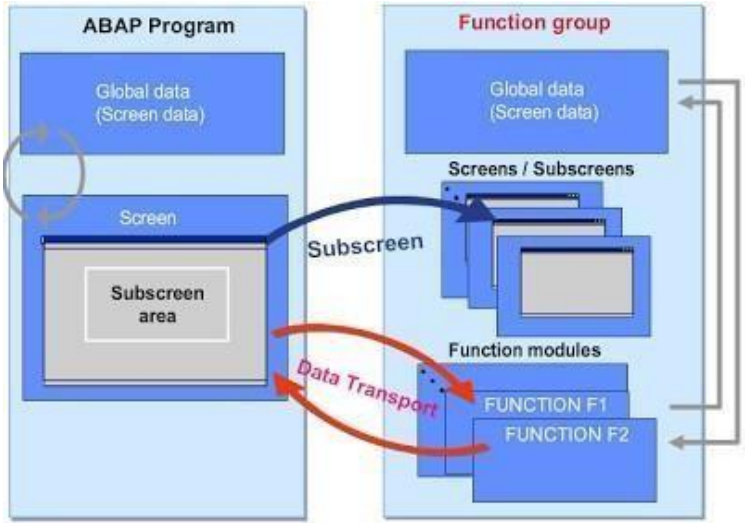


Fig. 5.8 Subscreens: Encapsulation in Function Group

There may be instances when you may need to use a single subscreen in the screens of several different programs. In such cases, you can encapsulate the subscreen in a function group and use function modules to transport data between the programs in which you want to use the subscreen and the function group. The interfaces of the function modules are used to pass the data between the calling program and the function group.

5.3.6 SUBSCREEN IN FUNCTION GROUP: CALL SEQUENCE

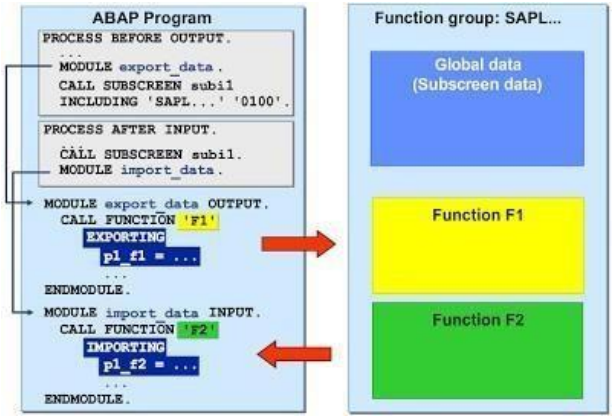


Fig. 5.9 Subscreen in Function Group: Call Sequence

You should use a module prior to the subscreen call to exchange data between the calling program and the function group. A function module is called for this purpose. It must be called before the subscreen call so that data is known in the function group before the PROCESS BEFORE OUTPUT processing block of the subscreen is called.

The PROCESS AFTER INPUT processing block of the subscreen is called before the function module call to transport the data back from the function group to the calling program.

5.3.7 SUBSCREEN IN FUNCTION GROUP: DATA TRANSPORT

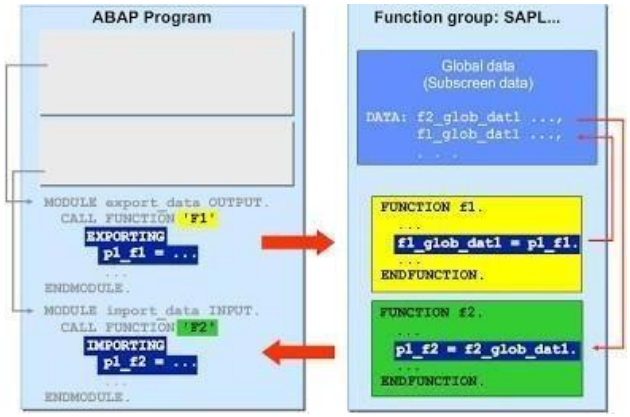


Fig. 5.10 Subscreen in Function Group: Call Sequence

You can access the data of the calling program globally in the function group by transferring the interface parameters from the function module into global data fields of the function group. In order to transfer data from the function group to the calling program, the corresponding data from the global data of the function group must be copied into the interface parameters of the function module.

5.3.8 SCREEN ELEMENT: TABSTRIP CONTROLS

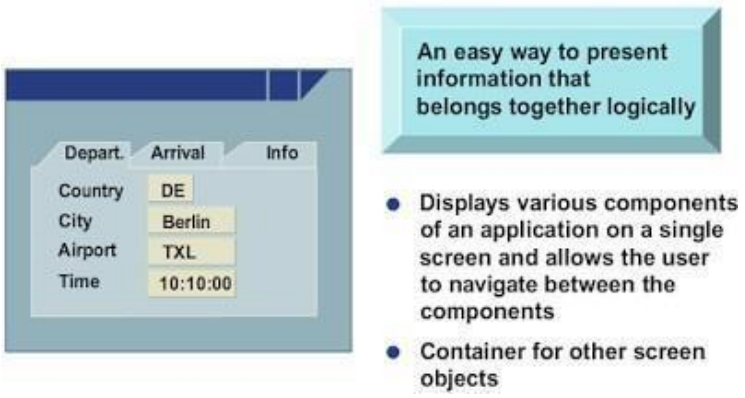


Fig. 5.11 Screen Element: TABSTRIP controls

Tabstrip controls serve as a container for other screen objects. They also provide an intuitive way of navigating between different components of an application shown on a single screen.

You can use tabstrip controls to depict different components of an application that form a logical unit.

5.3.8.1 TABSTRIP ELEMENTS

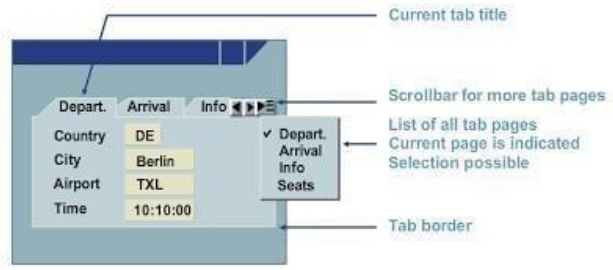


Fig. 5.12 TABSTRIP Elements

- A tabstrip control comprises individual pages with a tab page and a tab title.
- If the tabstrip control contains multiple pages, a scrollbar allows you to scroll through the different tab pages when it is not possible to display all the tab titles simultaneously. You can also use a pushbutton to display a list of all of the tab titles with a checkmark next to the active tab title.
- A tabstrip control also contains a border.

5.3.8.2 TABSTRIP CONTROLS: ATTRIBUTE

The general attributes, such as Object name, Starting position, and Static size, determine the name, position, and size of the tabstrip control, respectively. The resizable attributes can be used to check whether the size can be changed vertically and horizontally. You can also use the minimum size attribute to set a lower limit beyond which the size cannot be changed.

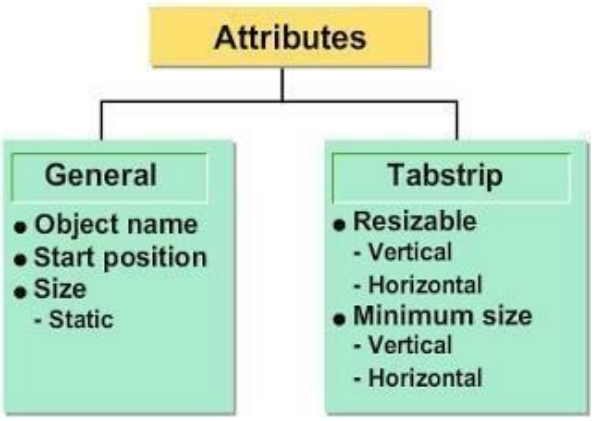


Fig. 5.13 Page Element: Technical View

5.3.9 PAGE ELEMENT: TECHNICAL VIEW

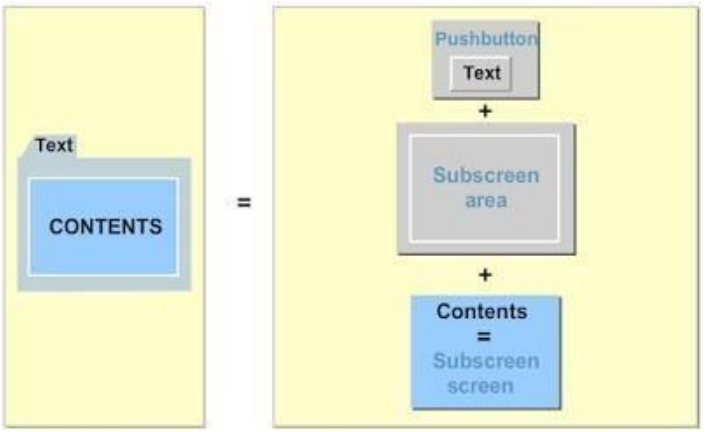


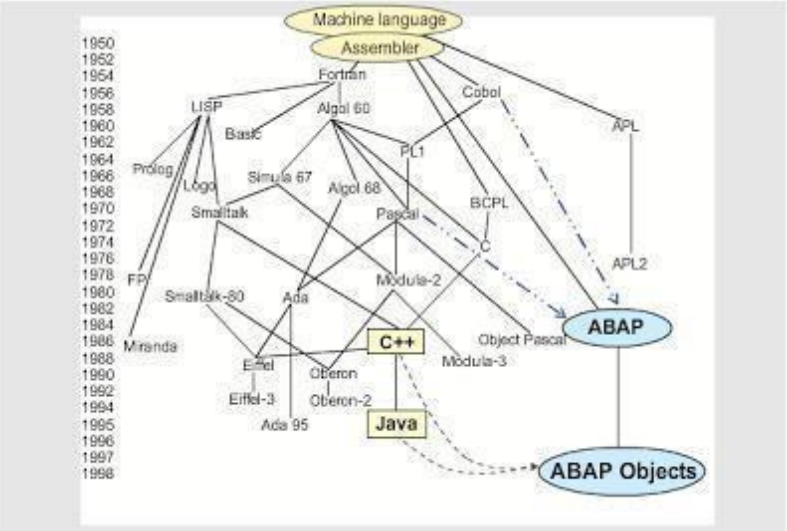
Fig. 5.14 Page Element: Technical View

A page element consists of a tab title, a subscreen area, and a subscreen. A tab title functions similar to a pushbutton. The subscreen technique is used to display the contents of the page elements. A subscreen area is assigned to each page element for which you can then call the subscreen.

CHAPTER 6

ABAP OBJECTS

6.1 INTRODUCTION TO OBJECT ORIENTED ABAP



- The main feature of Object Oriented programming is representing real-time objects in the form of class objects.
- Object Oriented ABAP focus on representing real-time objects of classes.
- SAP ABAP Object Oriented programming is available in two flavours.
- One is Global Classes and another one is local class.

Global Class

Global Class is an ABAP object which can be accessible via SAP Class Builder, T-code for SAP Class Builder is SE24.

Local Class

Local classes are classes which are available in ABAP programs, we can access them via ABAP editor SE38.

6.2 ABAP Classes and It`s components

What is a Class?

A class is a user defined data type with attributes, methods, events, user-defined types, interfaces etc for a particular entity or business application.

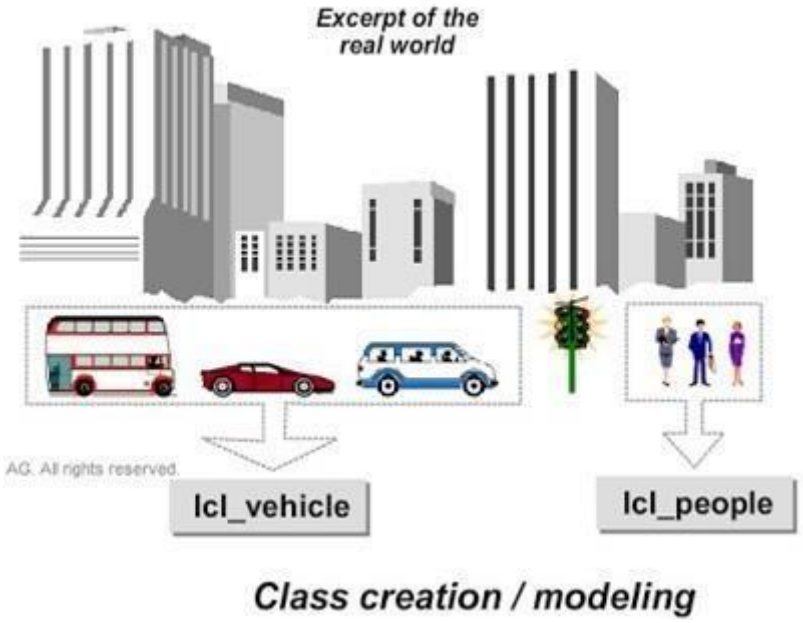


Fig. 6.1 Class creation / modelling

What are Objects?

Objects are nothing but instances of classes, each object has a unique identity that is memory and it`s own attributes.

Declaring Classes and Objects

Syntax: DATA TYPE REF TO.”Declaring global class in ABAP program using type ref t o(type reference to)

CREATE OBJECT. "Create object for the declared instance

Components of a Class

Attributes: Attributes are variables, constants declared within a class.

Methods: Methods are coding blocks which provides some functionality.

- These methods are similar to Function Modules in ABAP.
- Methods can access all attributes of it`s own class.
- Methods are defined in definition part and implemented in implementation part.
- We can call methods using `CALL METHOD` statement.

Events: Event is a mechanism through which one method of a class can raise method of other class, without hazard of instantiating that class.

Interfaces: Interfaces are similar to classes which contain methods without any implementation.

- Interfaces are mainly used to extend the scope or functionality of the class.
- Instance and Static Components

Instance components: These components exist separately in each instance (object) of the class and are referred using instance component selector using `->`

Static components: These components exists globally for a class and are referred to using static component selector `=>` .

6.2.1 Visibility of Components of Class

Each class component has a visibility.

In ABAP Objects, the whole class definition is separated into three visibility sections:

- PUBLIC.
- PROTECTED.
- PRIVATE.

Public section: Data declared in public section can be accessed by the class itself, by its subclasses as well as by other users outside the class.

Protected section: Data declared in the protected section can be accessed by the class itself, and also by its subclasses but not by external users outside the class.

Private Section: Data declared in the private section can be accessed by the class only, but not by its subclasses and by external users outside the class.

6.2.2 Global Class and Local Class

Classes in ABAP Objects can be declared either globally or locally.

Global Class: Global classes and interfaces are defined in the Class Builder (Transaction SE24) in the ABAP Workbench.

All of the ABAP programs in an R/3 System can access the global classes.

Local Class: Local classes are define in an ABAP program (Transaction SE38) and can only be used in the program in which they are defined.

6.2.3 COMPARISON OF CLASSES AND OBJECTS

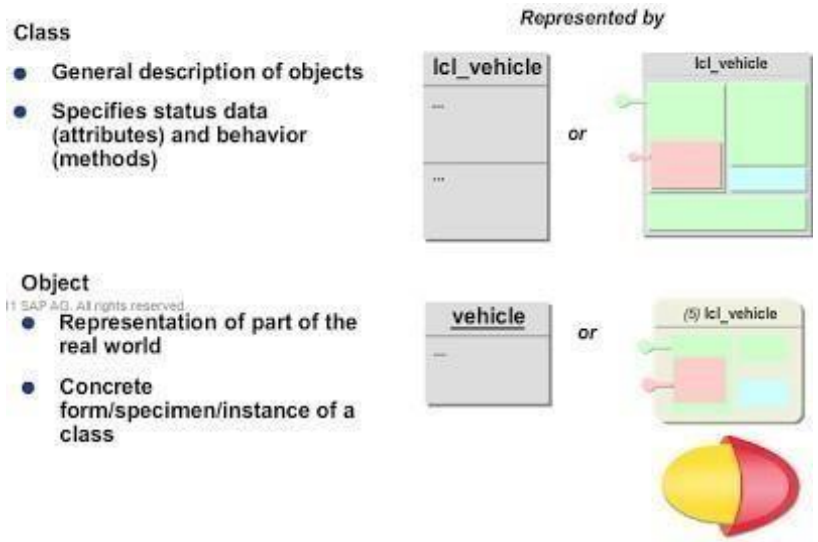


Fig. 6.2 Comparison of classes and objects

Let us put together the points to clearly understand the relationship between a Class and an Object. Class gives the general description of objects.

It specifies status data or attributes and behaviour or methods.

On the other hand, Object is a real time representation of the real world. It is a concrete form or specimen or instance of a class.

6.2.4 UML MODELING: DIAGRAM TYPES

Unified Modelling Language (or UML) is a globally standardized modelling language used for the specification, construction, visualization, and documentation of models for software systems. It enables uniform communication between users.

UML describes different diagram types to represent different views of a system. These three diagram types are of particular significance:

- Class diagrams represent the classes and the relationships between these in a static view of a model.
- Behaviour diagrams describe the sequence in which the objects are related to each other.
- Component diagrams depict the organization and dependencies of components.

6.2.4.1 REPRETATION OF A CLASS

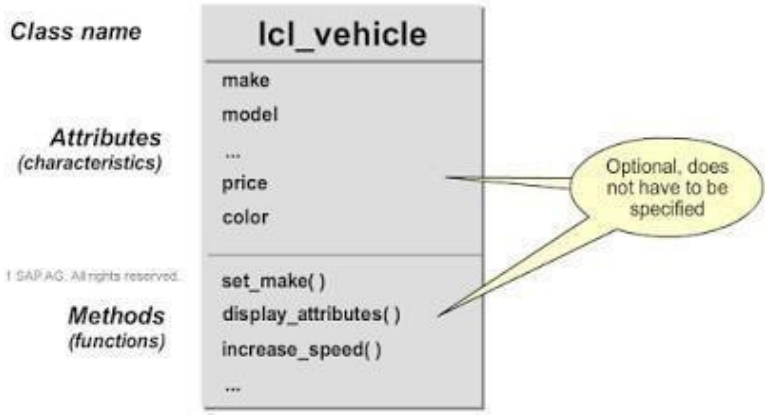


Fig. 6.3 Representation of a class in UML diagram

A class is represented by a rectangle in UML notation.

You first specify the name of the class, then its attributes, and finally its methods.
However, you can either omit both the attribute part and the method part, or just one of the two. Attributes define the data that can be stored in the objects of a class. They also determine the status of an object. Methods define the functions that an object can perform.

6.2.4.2 EXAMPLE OF A CLASS DIAGRAM

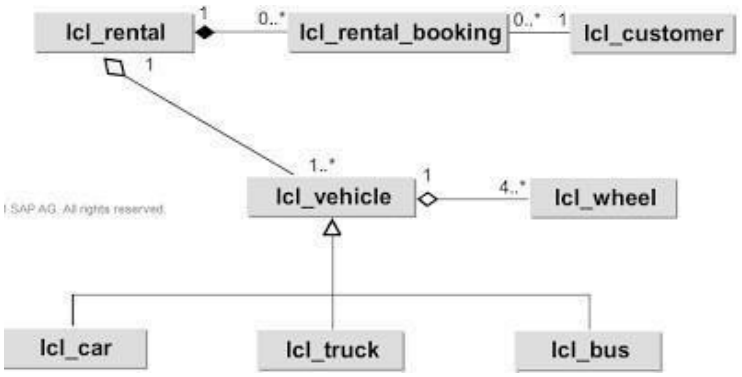


Fig. 6.4 Example of a class diagram

A class diagram illustrates all static relationships between the classes as shown here. There are two basic types of static relationships:

- Association
- Generalization or Specialization

You can understand association relationship through an example where a customer books a car at a rental car company.

You can understand generalization or specialization relationship through an example where a car, a bus, and a truck are all vehicles.

6.2.5 ASSOCIATION

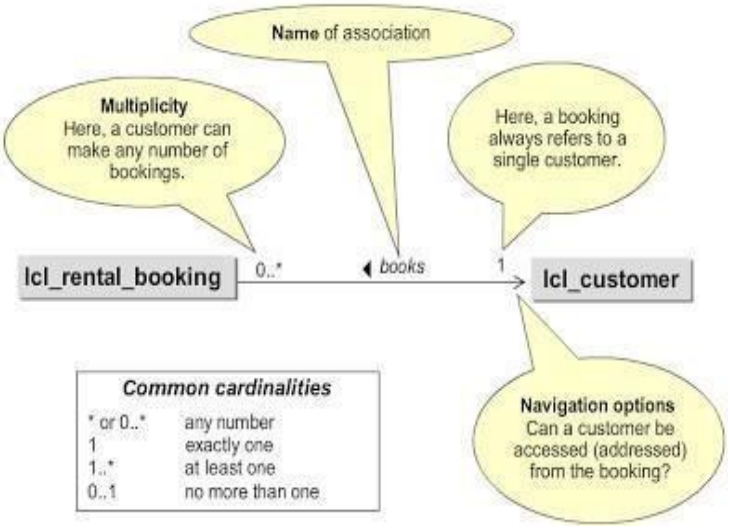


Fig. 6.5 Association

An association defines a semantic relationship between classes.

An association is represented by a line between the class symbols.

The cardinality, also referred to as multiplicity of the relationship, can be specified at each end of the line. The cardinalities generally used are

- 0…* which means any number;
- 1 which means exactly one;
- 1…* which means at least one; and
- 0…1 which means at most one.

Arrows indicate the navigation options, that is, the accessibility of the association partner.

An association can have a name, and is written in italics above the line. It may contain an arrow to show the direction in which it is to be read.

Each association has two roles, one for each direction of the association.

If roles are defined for both partners, role names can be entered at the end of the lines. Each role has a cardinality that shows how many instances can participate in this relationship. The multiplicity or cardinality is the number of participating objects in one class that have a relationship to an object in the other class.

In this example, a cardinality of "at least one" indicates that, only a person who actually makes a booking becomes a customer of the rental car company.

6.2.6 AGGREGATION AND COMPOSITION

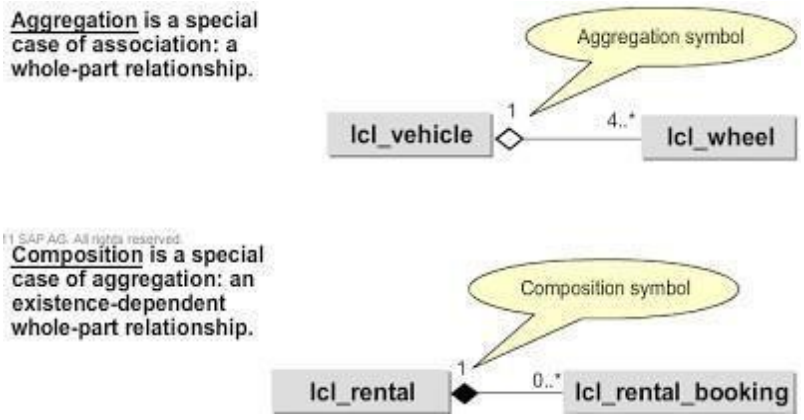


Fig. 6.6 Aggregation and Composition

Aggregation and composition are displayed as a line between two classes that is labelled with a small rhombus. Aggregation is a special case of association, a whole-part relationship. An empty rhombus is a symbol of aggregation. Composition is a special case of aggregation, based on whole-part relationship. Composition means that the contained object cannot exist without the aggregate. Therefore, the cardinality of the aggregate can only be one.

In UML notation, composition is denoted by a filled-in rhombus.

6.2.7 GENERALIZATION AND SPECIALIZATION

Generalization and specialization are specific relationships that exist between classes. These are always bidirectional.

Generalization or specialization relationships are indicated by a triangular arrow. However, this arrow always points towards the more general class. The extent of generalization increases in the direction of the arrow.

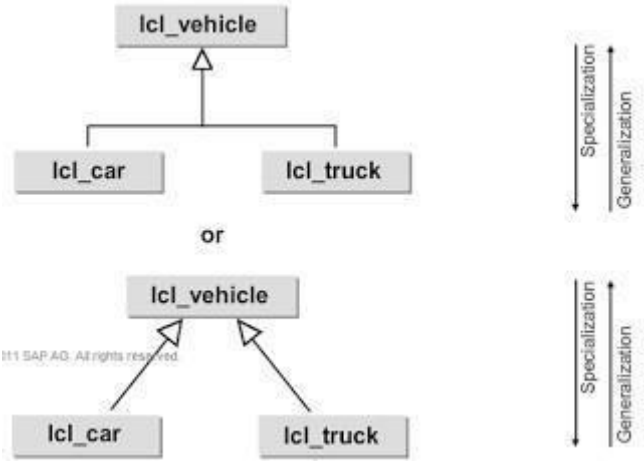


Fig. 6.7 Generalization and Specialization

6.2.8 OBJECT DIAGRAM

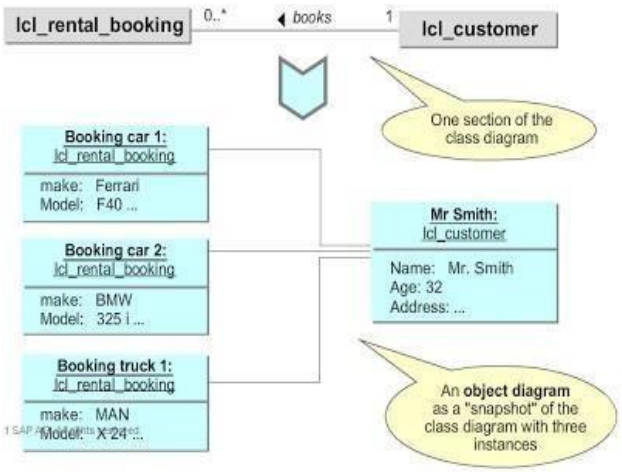


Fig. 6.8 Object Diagram

An object diagram illustrates the instances of the classes and the relationships between them. It can be defined as a variant of a class diagram which shows or depicts one section of the class diagram. It is useful for representing a complex class diagram.

In this example, an object diagram is represented as a “snapshot” of class diagram lcl_rental_booking with three instances Booking Car 1, Booking Car 2, and Booking Truck 1. All the three bookings here refer to a particular customer Mr. Smith of class lcl_customer. Attributes associated with the individual instances are also shown.

Messages are depicted as horizontal arrows between the object lines. The message is written above the arrow in the form of parameter as shown. The reply can be shown as a returning arrow.

6.3 DEFINING CLASSES

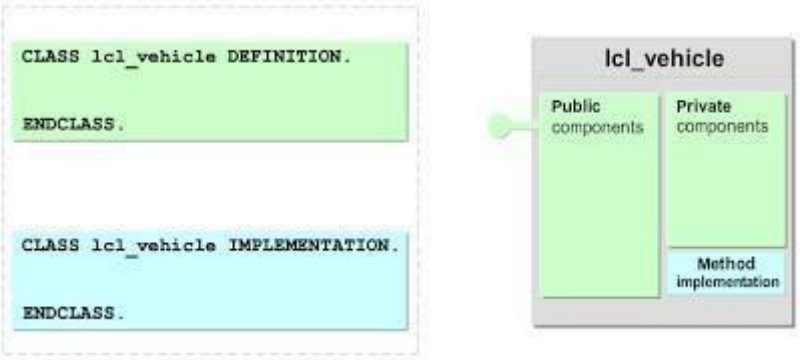


Fig. 6.9 Defining Classes

A class is a collection of objects that have similar structure and behaviour. In other words, a class acts as a blueprint on which all objects in that class are created. You can define a class using the keyword 'DEFINITION'. Generally, the syntax followed is CLASS class_name DEFINITION. Every class definition should end with keyword ENDCLASS.

All components of the class including attributes, methods, events, constants, and types are defined in the definition part.

A class also has an implementation part described by the keyword IMPLEMENTATION. Syntax is similar to that of definition part. Class implementation should also end with the keyword ENDCLASS. Only methods are implemented in the implementation part.

CHAPTER 7

ABAP ENHANCEMENTS

SAP is a ERP software in which all standard business applications are delivered, sometimes customer may need to alter existing functionality or add additional functionality to the existing applications based on customer business requirements.

7.1 Enhancements concept

The enhancement is a concept of adding your own functionality to SAP's standard business applications without having to modify the original applications. To modify the standard SAP behavior as per customer requirements, we can use enhancement framework.

Types of enhancements in SAP

User Exits

Initially SAP implemented enhancements in the form of User Exits and these are only available in SD module, user exits are implemented in the form of subroutines and hence are also called as FORM EXITS, User exits are empty subroutines that SAP developers have provided for you, you can add your own source code in the empty subroutines.

All user exits starts with the word USEREXIT.

Customer Exits

These are one type of enhancements that are available in some specific programs, screens and menus within standard SAP Applications. These are Function Modules with a custom empty include program, you can add your own functionality in these include programs.

Customer Exits are available in all SAP modules, whereas user exits are only available in SD module.

All customer exits (Function Modules) starts with CALL CUSTOMER word.

7.1.1 Enhancement Framework

Purpose

The new enhancement concept of the ABAP Workbench (Enhancement Framework) enables the integration of different concepts for modifying and enhancing development objects. The enhancement concept is supported by the Enhancement Builder tool and ABAP language elements.

Previous Concepts

In the long-term, the new Enhancement Framework is to replace or incorporate the existing enhancement and modification concepts. These are described under Changing the SAP Standard (BC).

The previous concepts were separated into modifications – that is, changes to delivered development objects, and enhancements – that is, inserting user developments at points predefined by SAP.

- Modifications were carried out using the Modification Assistant and supported in system upgrades.
- Before Release 4.6, enhancements were only possible via Customer Exits. As of
- Release 4.6, function module exits (customer exits for source code enhancements) were replaced with Business Add-Ins.

  Problematic aspects of the previous concepts are:

- There is no real upgrade support for modifications and enhancements made in different systems of a transport route – for example, by SAP, to partner add-ons, or to customer developments.
- It is difficult to trace developments made in different parallel systems back to one system.
- In the case of systems in which large amounts of user developments and modifications were made, retracing and testing these developments after an upgrade is a very time-consuming process.

7.1.2 Features

The aim of the new enhancement concept as of Release 7.0 of the SAP NetWeaver Application Server ABAP (SAP NetWeaver 7.0) is to unify all possible ways of modifying or enhancing SAP products (more precisely, Repository objects of the SAP

NetWeaver Application Server ABAP), which go beyond the scope of Customizing. The corresponding tool is the Enhancement Builder, which is integrated in the ABAP Workbench.

In the context of the new enhancement concept, the following can be handled as enhancements:

- Modifying a Repository object
- Replacing a Repository object with an identically-named object
- Enhancing a Repository object at a predefined position
- Using a foreign object

The Enhancement Framework knows all information about an enhancement that is required, for example, for an upgrade or for the ABAP runtime environment.

The enhancements of the enhancement concept can be switched using the Switch Framework. This means that an enhancement takes effect when the package in which the above enhancement components are defined is assigned to a switch of the Switch Framework and this switch is not deactivated.

7.1.3 Multilayer Support

In contrast to modifications, with enhancements it becomes possible to have enhancements on different development levels, for example:

- core development
- application development
- add-on development
- customer development

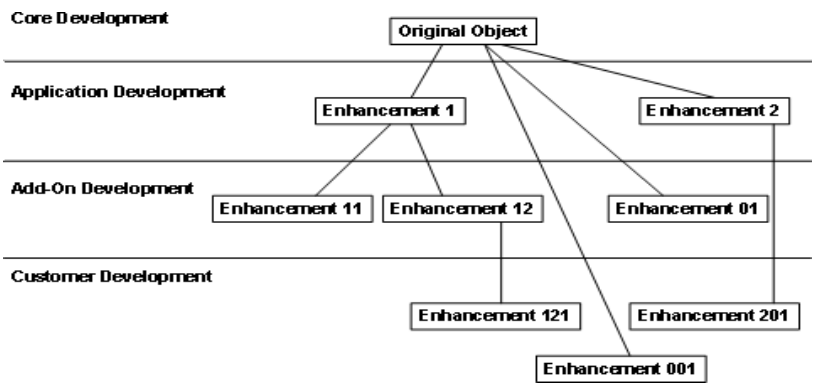


Fig. 7.1 Different development levels

It is possible to create multiple enhancement implementations on different layers or to replace an enhancement implementation, but enhancements cannot be nested.

7.1.4 ENHANCEMENT CONCEPT (OVERVIEW)

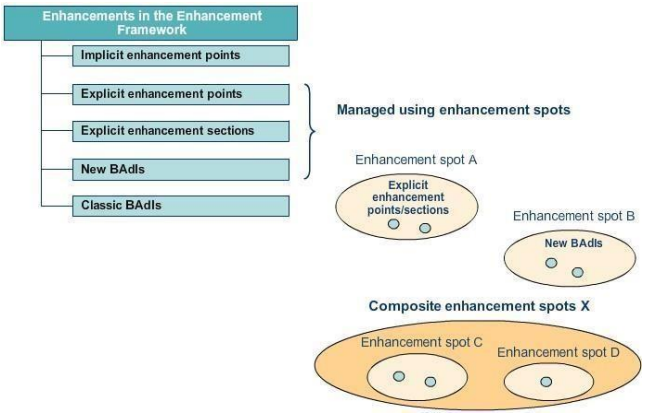


Fig. 7.2 Enhancement Concept

The new enhancement framework contains enhancement points and enhancement sections as newly added options. It includes both implicit and explicit enhancement points and explicit enhancement section. The new enhancement framework also contains new BAdI and classic BadI. As shown here, explicit enhancement point, explicit enhancement section and the new BadIs are managed using enhancement spots. A collection of enhancement spots can be called a composite enhancement spot.

7.1.5 ENHANCEMENT POINT

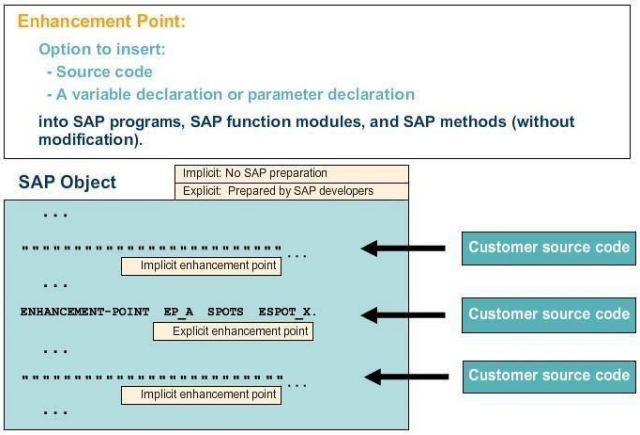
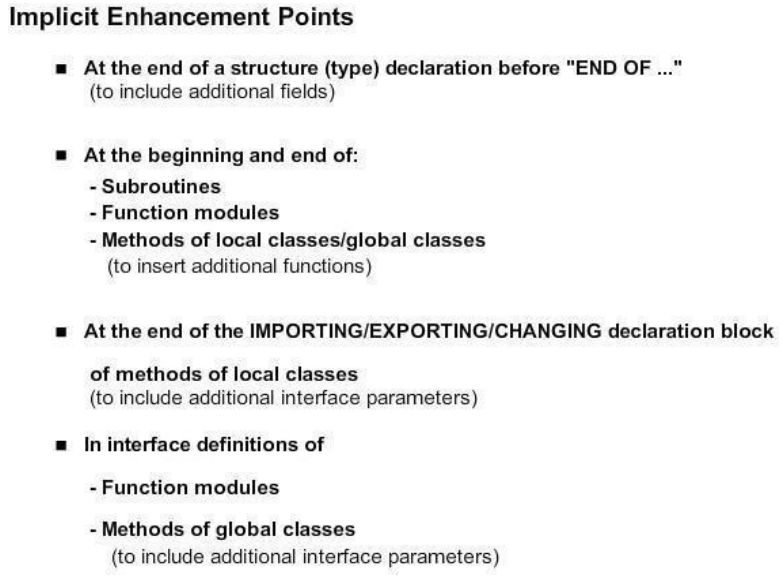


Fig. 7.3 Enhancement Point

An enhancement point provides the users with an option to insert their own source code, variable declarations, and parameter declarations in SAP programs, function modules and classes without the need for a modification. Implicit enhancement points are present at particular points in SAP objects by default, whereas explicit enhancement points are insertion options that are generated by SAP.

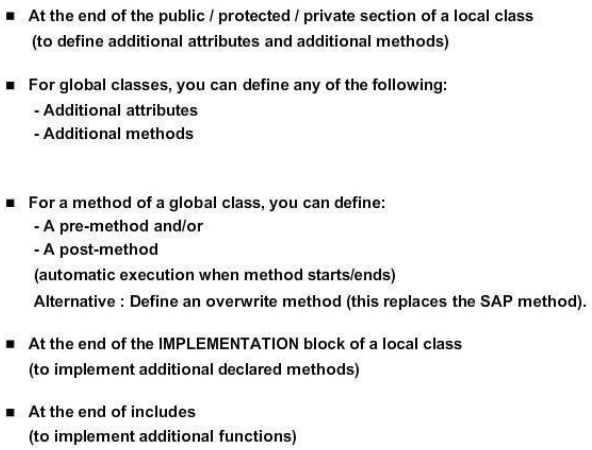
Composite enhancement spot has many enhancement spots Enhancement spot is what the user creates implementations from Enhancement spot contains many enh points and many enh sections

7.1.6 IMPLICIT ENHANCEMENT POINTS (1)



The slide lists the options where implicit enhancement points are present.

7.1.7 IMPLICIT ENHANCEMENT POINT (2)



As shown here, implicit enhancement points are also present in various points in classes.

7.1.8 EXPLICIT ENHANCEMENT POINTS AND ENHANCEMENT SECTIONS

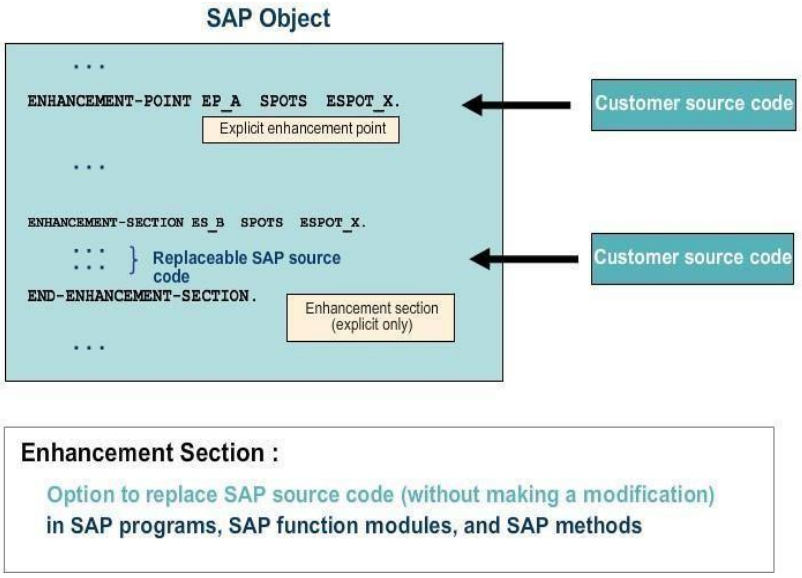


Fig. 7.4 Explicit enhancement points and enhancement sections

An explicit enhancement point is an option provided in advance to SAP users to enhance the SAP source code without making any modification.

Similarly, an explicit enhancement section is another such option which allows the users to replace the SAP source code without making any modification.

Enhancement point can be either static (to allow additional data declaration) or dynamic (to allow additional executable code).

Both point and section are stored inside a spot.

Enhancement Point -

If you have written code using enhancement point your custom code will be executed along with the standard code.

Enhancement Section -

If you have written code using enhancement section, only your custom code will be executed replacing standard code. Standard code will not be executed.

7.2 User Exits in SAP

These are implemented in the form of subroutines and hence are also known as **FORM EXITs**. The user exits are generally attached to the standard program by the SAP.

User exits are a type of system enhancement that was originally developed for the R/3 SD (Sales and distribution) module.

User-exits are empty subroutines that SAP Developers have provided for you.

You can fill them with your own source code. Technically this is a modification.

All User exits start with the word **USEREXIT_**...FORM

USEREXIT_XXXX... ENDFORM.

7.2.1 User exits can be found in the following ways:

Go to Object Navigator (SE80), select Package and put VMOD (Application development R/3 SD customer modification) and press enter. You will find all the includes for user exits in SD. You will get User exits for Sales order, Delivery, Billing, Pricing etc. Most of the time documentation is maintained for each subroutine which helps developer for better understanding of the subroutine. Select the subroutine according to the requirement and start coding within the subroutine.

Examples:

-In User exits MV45AFZZ(Sales Order Exit), we have subroutine

USEREXIT_PRICING_PREPARE_TKOMK

USEREXIT_PRICING_PREPARE_TKOMP

This user exit can be used to move additional fields into the communication table which is used for pricing. TKOMK for header fields and TKOMG for item fields. The fields which are not in either of the two tables KOMK and KOMP cannot be used in pricing.

-In User exits MV50AFZ1(Delivery Exit), you have subroutine

USEREXIT_SAVE_DOCUMENT_PREPARE

This user exit can be used for changes or checks, before a document is saved.

7.2.2 Customer Exits in SAP

SAP creates customer exits for specific programs, screens, and menus within standard applications. These exits do not contain any functionality. Instead, the customer exits act as hooks. You can hang your own add-on functionality onto these hooks.

Customer exits are nothing but a include in customer name space will be provided in the function module which starts with CALL CUSTOMER. You can fill them with your own source code. Technically this is an enhancement. User exits generally refer to SD module while customer exits refer to all modules like MM, SD, PP, FICO etc.

Advantages of Customer Exits:

- They do not affect standard SAP source code.
- They do not affect software updates**.**

Disadvantages of Customer Exits:

- Customer exits are not available for all programs and screens found in the SAP System. You can only use customer exits if they already exist in the SAP System.

Types of Customer Exits

1. Function Module exits.
2. Screen exits.
3. Menu exits.

1. Function Module exits.

Function module exits are exits developed by SAP. The exit is implemented as a call to a function module. The code for the function module is written by the developer. You are not writing the code directly in the function module, but in the include that is implemented in the function module.

Format: CALL CUSTOMER-FUNCTION '910' The naming standard of function modules for function module exits is:

```
EXIT_<3 digit suffix>.
```

Examples: Function Module Exits: EXIT_SAPMF02K_001.

2. Screen Exits:

Allow customer to add fields to a screen via a sub screen in an SAP program. The sub screen is called within the standard screen's flow logic. For this we need to implement FM Exits/Menu Exits to write the logic for Screen Exits. All screen exits will be subscreens.

3. Menu exits:

Menu exits allow you to add your own functionality to menus. Menu exits are implemented by SAP and are reserved menu entries in the GUI interface. The developer can add his/her own text and logic for the menu.

Function codes for menu exits all start with "+". Format: +CUS (additional item in GUI status)

7.2.3 Methods to find out customer Exits:

There are numbers of way by using we can find out Customer Exits.

Enhancement: It is a group or container of the FM exits/Menu Exits/Screen Exits .

Project: It is an SAP Object which contains list of Enhancements. A project must be created for each enhancement so that the corresponding exits in an enhancement will be in active state. So, A project is group of Enhancements An Enhancement cannot be linked to more than one project. It throws error CMOD is the Tcode for creating a project.

Finding customer exit with CALL CUSTOMER

We know every transaction is linked to a program, all customer exits(Function Modules) inside programs starts with CALL CUSTOMER, so we can use CALL CUSTOMER to find them.

CHAPTER 8

WEB DYNPRO

8.1 INTRODUCTION

Web Dynpro for ABAP or Web Dynpro for ABAP (WD4A, WDA) is the SAP standard UI technology for developing Web applications in the ABAP environment. It consists of a runtime environment and a graphical development environment with special Web Dynpro tools that are integrated in the ABAP Workbench (SE80).

Web Dynpro offers the following advantages for application developers:

- The use of declarative and graphical tools significantly reduces the implementation effort
- Web Dynpro supports a structured design process
- Strict separation between layout and business data
- Reuse and better maintainability by using components
- The layout and navigation is easily changed using the Web Dynpro tools
- Stateful applications are supported – that is, if the page is changed and the required data remains intact so that you can access it at any time throughout the entire application context.
- Automatic data transport using data binding
- Automatic input check
- User interface accessibility is supported
- Full integration in the reliable ABAP development environment

8.2 APPLICATION SCENARIOS WITH WEB DYNPRO

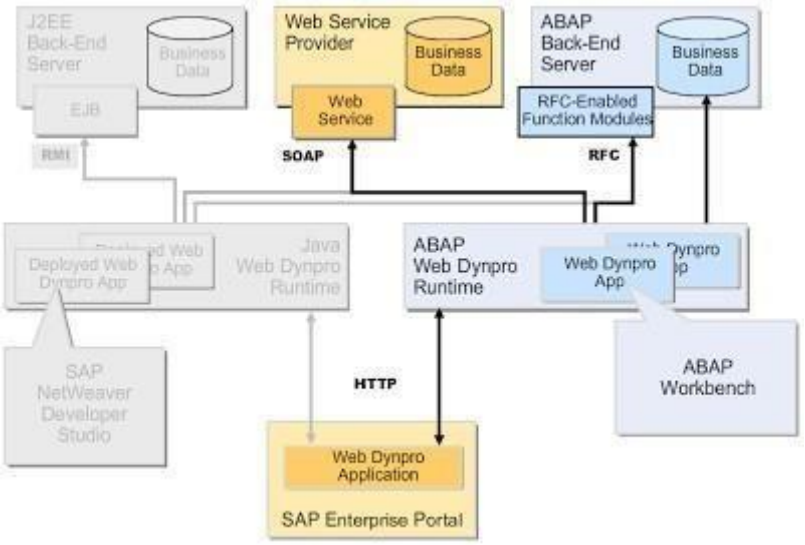
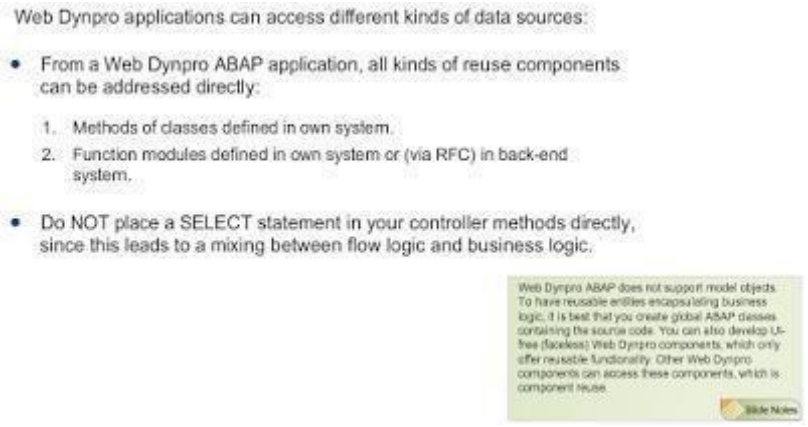


Fig. 8.1 Application Scenarios with Web Dynpro

SAP Enterprise Portal comprises Web Dynpro Application, through which we can access ABAP Web Dynpro Runtime. Web Dynpro applications are created using ABAP Workbench. ABAP Back-End Server processes these applications. RFCs work through RFC-enabled function modules to pass data into the ABAP back-end server, where it is stored as business data. The Web service also carries this data to the Web service provider through SOAP, where it is stored as Business Data.

8.3 WEB DYNPRO APPLICATION: DATA SOURCES


Web Dynpro applications can access different kinds of data sources:

- From a Web Dynpro ABAP application, all kinds of reuse components can be addressed directly:
  1. Methods of classes defined in own system.
  2. Function modules defined in own system or (via RFC) in back-end system.
- Do NOT place a SELECT statement in your controller methods directly, since this leads to a mixing between flow logic and business logic.

Fig. 8.2 Web Dynpro Application

Web Dynpro applications can access different kinds of data sources:

- You can address all kinds of reuse components directly from a Web Dynpro ABAP application: Methods of classes that are defined in your own system
- Function modules that are defined in your own system or in a back-end system using RFC Web Services through a Web Service client object.

Placing a SELECT statement in your controller methods directly leads to a mixing between flow logic and business logic.

8.4 WEB DYNPRO BENEFITS

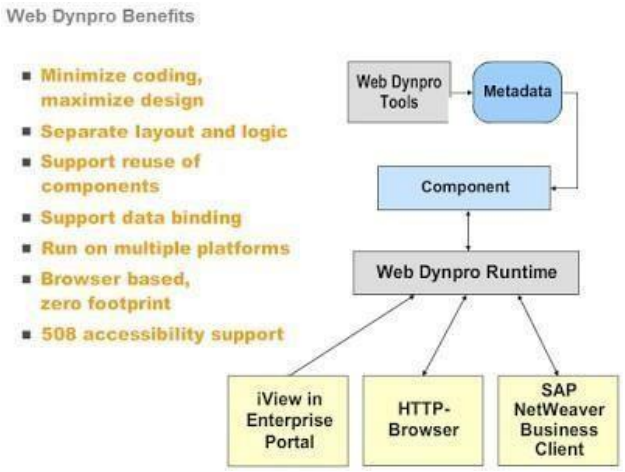


Fig. 8.3 Web Dynpro Benefits

Web Dynpro uses descriptive tools in a structured design process to enable application developers to create powerful Web applications with minimal effort. It defines user interfaces through a declarative meta model. We can list some other advantages of Web Dynpro:

- Minimized coding and maximized design
- Separate layout and logic Reusability of components Binding of data
- Availability of multiple platforms
- Browser-enabled manipulation with zero footprint
- 508 accessibility support

8.5 WEB DYNPRO COMPONENT

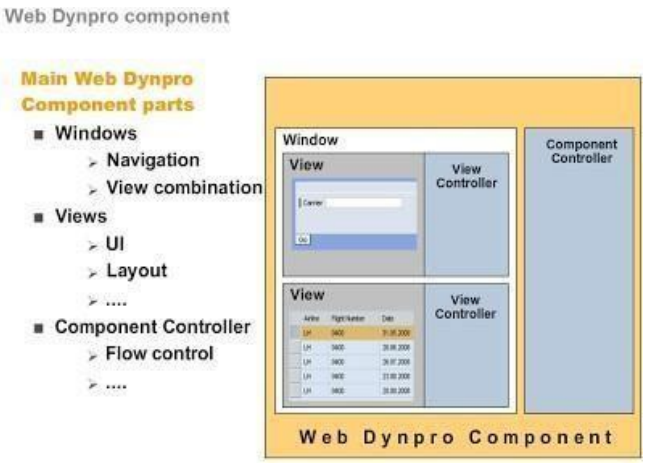


Fig. 8.4 Web Dynpro Component

Web Dynpro Component houses the entities related to the user interface (UI) and Web Dynpro. We can list the main Web Dynpro components, which are entities related to the UI:

- **Windows** define the possible combinations of views. Windows can contain any number of views and a view can also be embedded in any number of windows. Windows also allow for the flow between views and the necessary navigation.
- **Views** comprise UI elements, such as input fields and buttons, as well as view layouts, such as the rectangular part of a page or a browser displayed by the client. The client can set up a complete page by using just one view or multiple views. View controllers aid these manipulations.
- **Component controllers** help maintain flow control between all the components of the Web Dynpro.

8.6 CONTEXT AND DATA TRANSPORT

A context is a hierarchical data storage system, which stores data related to the UI.

User input values for UI elements are connected to the context attributes of the corresponding view controller. Through data binding, automatic data transport between the UI elements and the context attributes is possible.

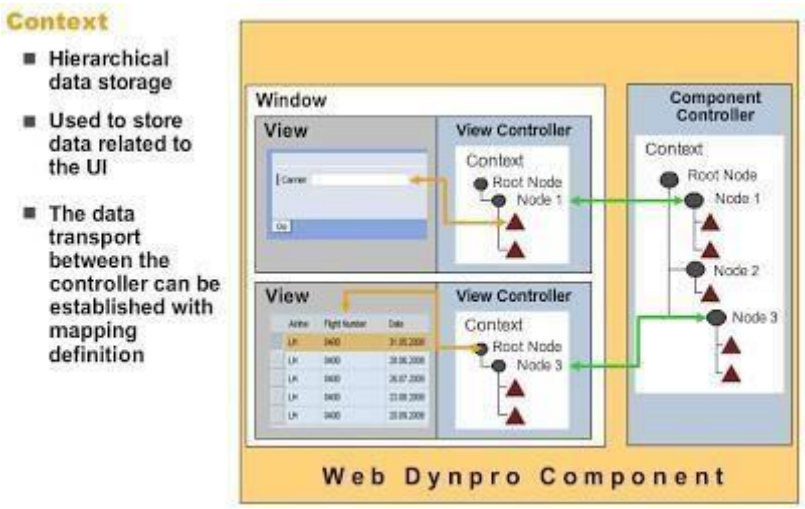


Fig. 8.5 Context and Data Transport

You can reference the variables defined in a Web Dynpro controller from other Web Dynpro controllers through context mapping.

Combining the two concepts – context mapping and data binding – data transport between UI elements located in different views can be defined in a purely declarative way.

8.7 CONTENT MAPPING

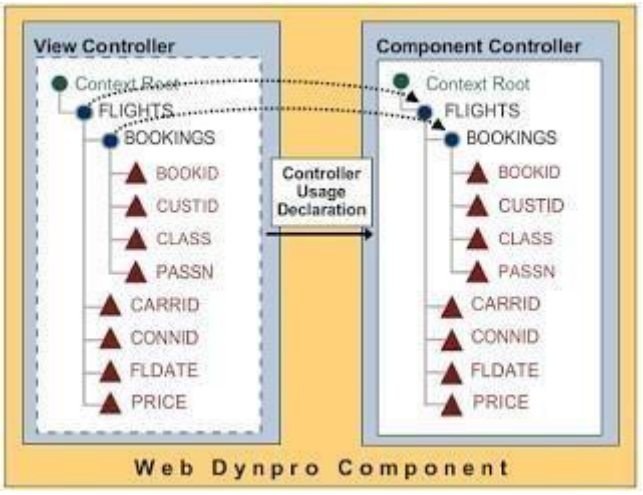


Fig. 8.6 Context Mapping

A context node in one controller can automatically access data from a context node in another controller using context mapping.

The controller acting as the mapping origin contains the relevant node in its context, This node also has child nodes or attributes.

The mapping origin controller must not be a view controller.

The controller containing the mapped node must declare the use of the mapping origin controller as a used controller.

8.8 PUTTING DATA ON SCREEN: DATA BINDING

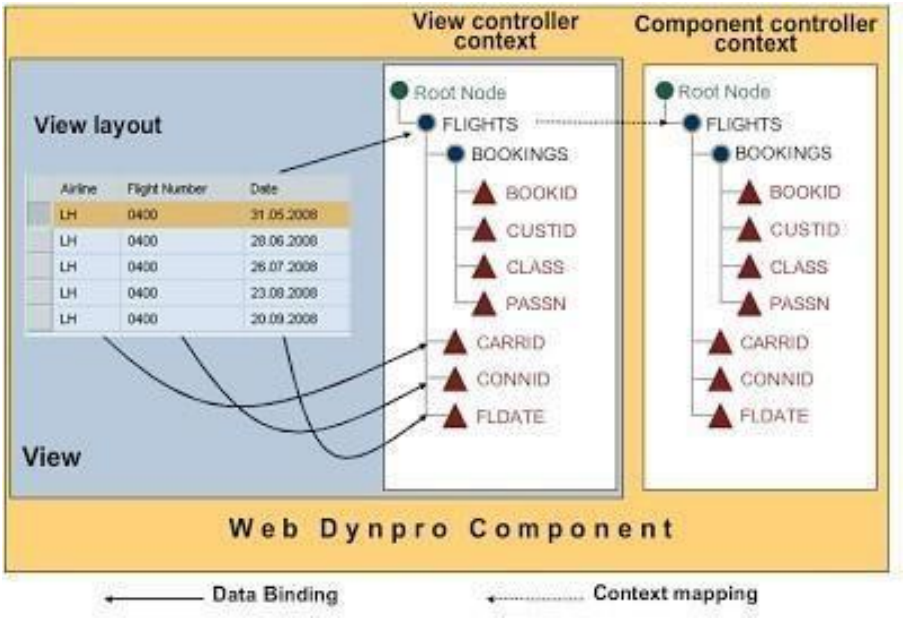


Fig. 8.7 Putting Data on Screen: Data Binding

Data binding helps transport data automatically from a view controller's context to a UI element in its layout, and vice versa.

UI elements, being private to the view controller in which they are declared, cannot be bound to context nodes or attributes defined in another controller.

- The Web Dynpro framework performs two tasks, subsequent to databinding: It transports data from the context attribute to the UI.
- It repopulates the context attribute from the UI element after the user enters the data and initiates the next server round trip.

8.9 NAVIGATION: PRINCIPLE

Navigation links join outbound and inbound plugs together.

An inbound plug is linked to an outbound plug by registering the inbound plug event handler method to the navigation event called by firing an outbound plug.

Navigation links are defined in a window.

An Action is used to link a client side event to an event handler method (automatically defined with the action) in the corresponding view controller.

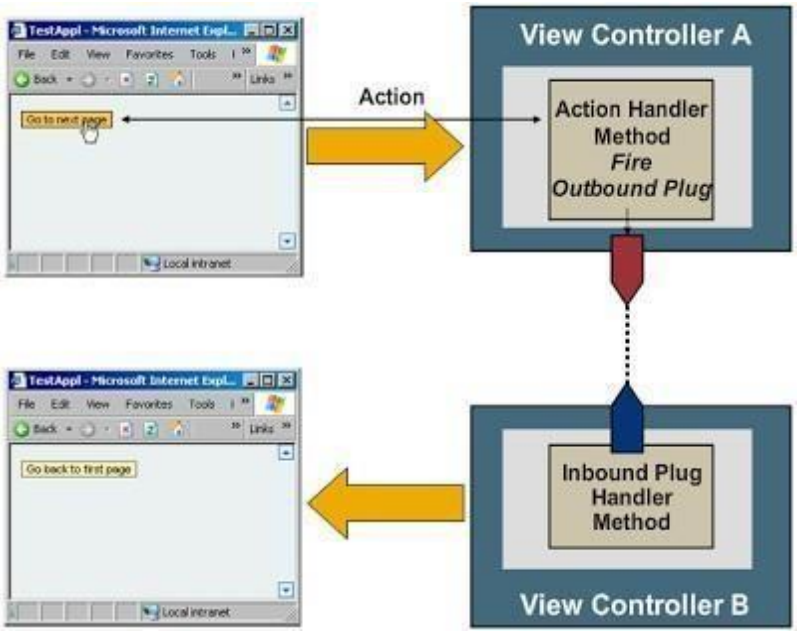


Fig. 8.8 Navigation Principle

8.10 NAVIGATION BETWEEN VIEWS

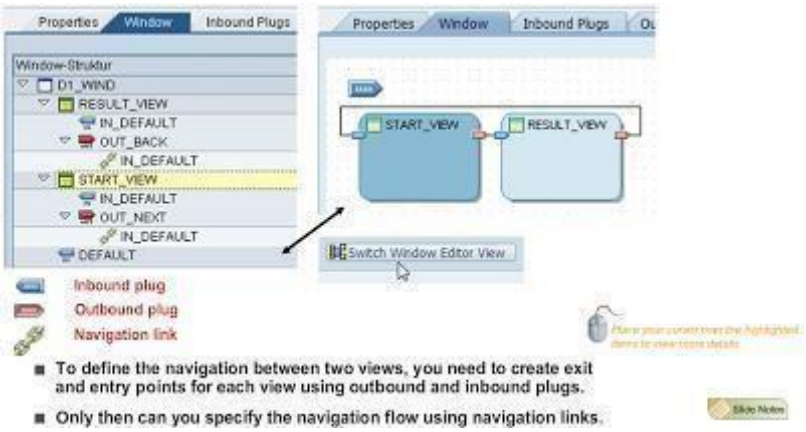


Fig. 8.9 Navigation between views

You can define the navigation between two views by creating exit and entry points for each view using outbound and inbound plugs.

You can then specify the navigation flow using navigation links.

8.11 MODEL VIEW CONTROLLER

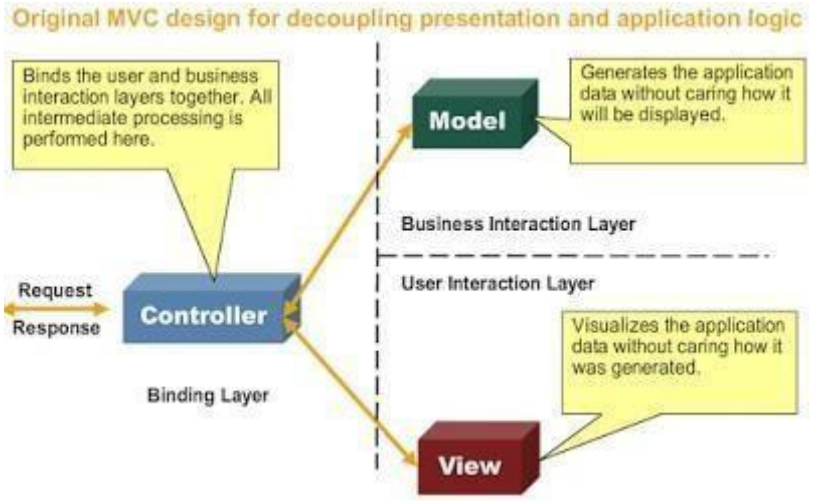


Fig. 8.10 Model View Controller

The **Model View Controller** (MVC) design paradigm is the foundation for SAP's Web Dynpro. It decoupled presentation and application logic, using three layers:

- The **Binding Layer** binds the user and interaction layers together. The Controller, which is at the heart of this layer, performs all intermediate processing.
- The **Business Interaction Layer** consists of the Model, which generates application data without caring how it will be displayed.
- The **User Interaction Layer** or **View** visualizes the application data without caring how it was generated.

8.12 INTERNALLY VISIBLE WEB DYNPRO ENTITIES (1)

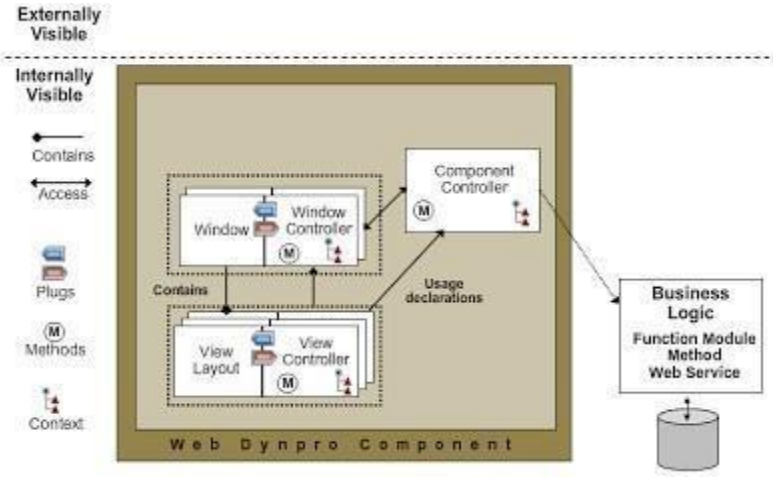


Fig. 8.11 Internally Visible Web Dynpro Entities (1)

- Two types of entities make up a Web Dynpro component – those that are **externally visible** and those that are **internally visible**.
- The internally visible parts can be either visual or programming entities, which consist of windows and views.
- A view consists of a view layout and the corresponding view controller. The view controller can contain navigation plugs, methods, and a context.
- A window embeds one or more views and has a corresponding window controller. A window controller can contain navigation plugs, methods, and a context.
- A view can be embedded in different windows, the first view so embeddedbeing the default view.

8.13 INTERNALLY VISIBLE WEB DYNPRO ENTITIES (2)

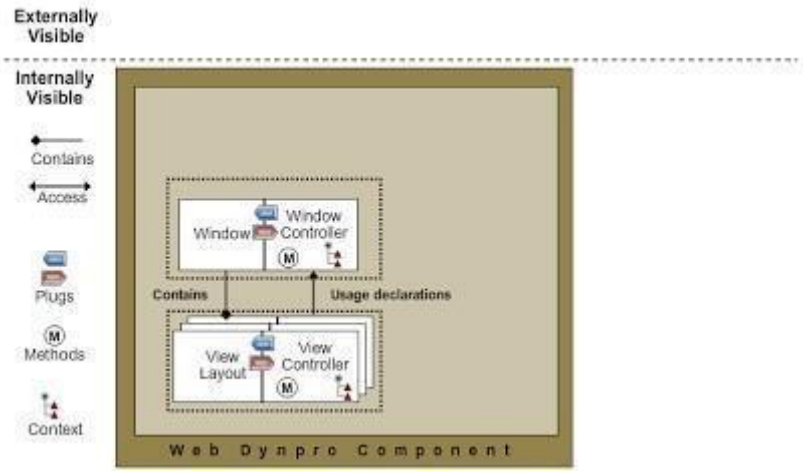


Fig. 8.12 Internally Visible Web Dynpro Entities (2)

Business Logic is not part of the Web Dynpro component. It is defined outside, so that it has high reusability. Global ABAP classes are preferred for encapsulating the related source code.

The component controller acts as a component wide controller.

In the related view controller, only the program logic related to a certain view, such as checking user input, is coded.

8.14 INTERNALLY VISIBLE WEB DYNPRO ENTITIES (3)

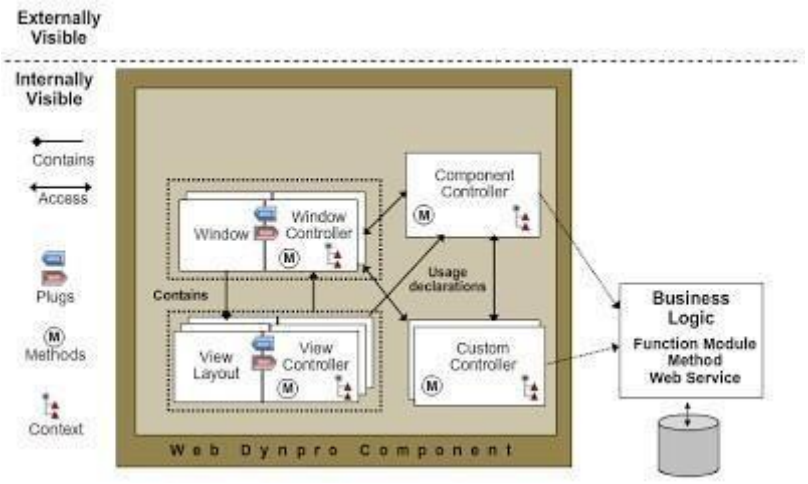


Fig. 8.13 Internally Visible Web Dynpro Entities (3)

The developer defines optional controllers, which then become custom controllers. They help modularize component content. They can function as local controllers for some views. Alternatively, they help encapsulate the logic related to a certain model class. By creating a usage declaration in the custom controller for the component controller and vice versa, you can reduce the content of the component controller and populate sub functions.

8.15 VISIBLE WEB DYNPRO ENTITIES

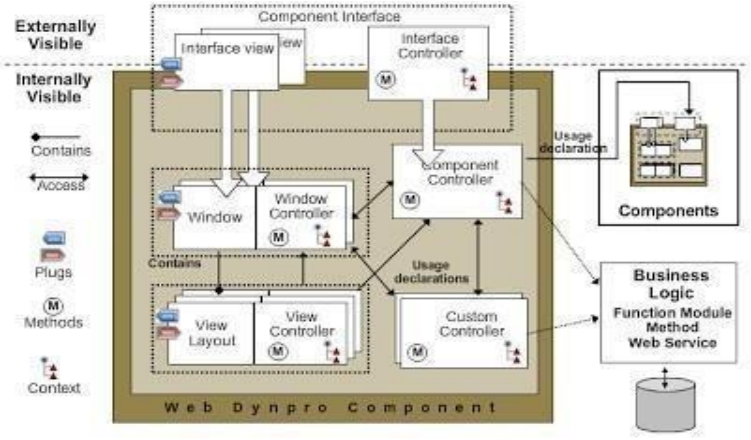


Fig. 8.14 Visible Web Dynpro Entities

For one Web Dynpro component (parent component) to access another (child component), the former can declare the use of the latter. A specific component usage instance is then created. Through the component interface controller, the parent component accesses the functionality of the child component.

The externally visible entities of the Web Dynpro component are the interface controller and the interface view(s).

8.16 WEB DYNPRO APPLICATION

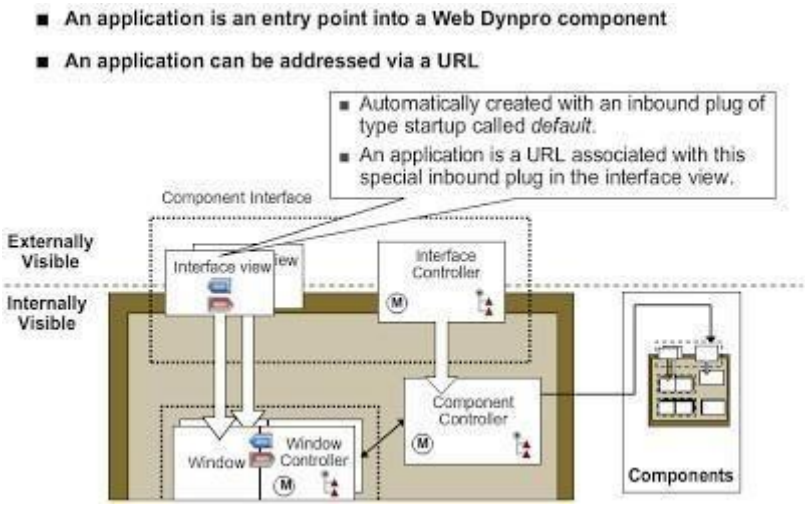


Fig. 8.15 Web Dynpro Application example

A Web Dynpro application is the only Web Dynpro entity that can be addressed via a URL, and is an entry point into a Web Dynpro component.

The Interface view in Component Interface has inbound as well as outbound plugs. An application is created here with an inbound plug of the type Startup, called Default. This is because an application is a URL associated with this special inbound plug in the interface view.

8.17 WEB DYNPRO APPLICATION: TYPES OF CONTROLLERS

- **Component controller**
  There is only one component controller per Web Dynpro component. This is a global controller, visible to all other controllers. The component controller drives the functionality of the entire component. This controller has no visual interface.
- **Custom controllers**
  Custom controllers are optional. They have to be defined at design time and can be used to encapsulate sub-functions of the component controller. Multiple custom controllers can be defined in a component. Custom controllers are instantiated automatically by the Web Dynpro framework and the instantiation order is undefined; therefore, the coding in a custom controller should not depend on the existence of any other custom controller.
- **Configuration controllers**
  This is a special custom controller. It is only necessary if the corresponding component implements special configuration and personalization functions. Only one configuration controller may exist in any component. Any controller can access the configuration controller, but the configuration controller cannot access any other controller.
- **View controllers**
  Each view consists of the layout part and exactly one view controller. This controller handles the view-specific flow logic, like checking user input and handling user actions.
- **Window controllers**
  Each window has exactly one window controller. This controller can be used to handle the data passed via the inbound plugs when being reused as a child controller. Methods of this controller can be called from the inbound plug methods of the window.

Four main types of controllers exist in a Web Dynpro component:

The **Component controller** drives the functionality of the entire component. It is a global controller; is visible to all other controllers; and has no visual interface.

- **Custom controllers** are optional. They are defined at design time and are used to encapsulate sub- functions of the component controller. A component can have several defined custom controllers, which are instantiated automatically by the Web Dynpro framework.
- The **Configuration controller** is a special custom controller. It is used for special configuration and personalization functions. A component can have only one configuration controller.
- The **View controller** handles the view-specific flow logic, such as checking user input and handling user actions. Each view comprises the layout and exactly one view controller.
- The **Window controller** handles the data passed through the inbound plugs when reused as a child controller. Each window has exactly one window controller. Its methods are also called from the inbound plug methods of the window.

8.18 CONSTITUENTS OF ALL CONTROLLERS

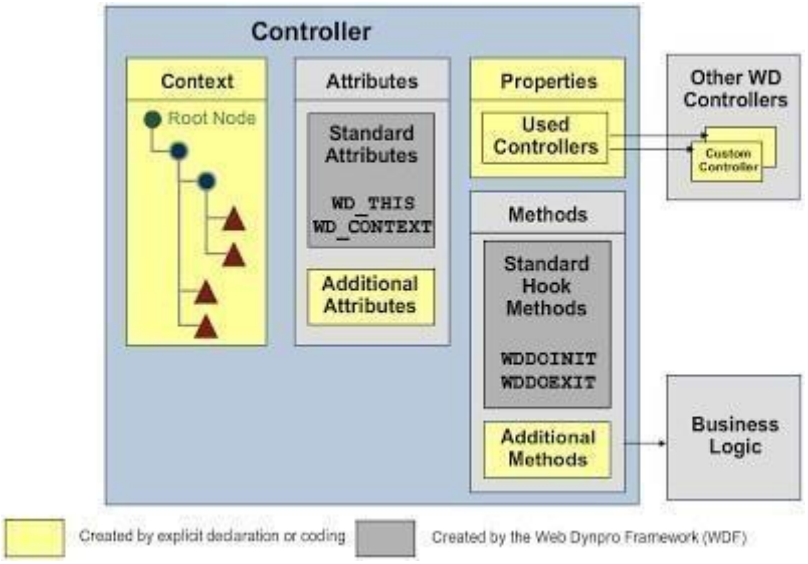


Fig. 8.16 All Controller Constituents

In the Web Dynpro framework, the Controller comprises Context, Attributes, Properties, and Methods. The Context houses all the root nodes.

Attributes comprise Standard Attributes and Additional Attributes. Standard attributes may be of two predefined types: WD_THIS and WD_CONTEXT, which access the functionality of the controller and the context, respectively.

The Properties tab of a controller defines the sharing of information between different controllers. This is where usage declarations are stored. Any controller can get access to another only by using this tab.

The Methods tab is the repository for hook methods, which are a set of methods predefined in the Web Dynpro framework. All controller types have at least two hook methods, which are processed only once during the lifetime of a controller instance.

WDDOINIT( ) is processed when a controller instance is created and WDDOEXIT( ) when a controller instance is deleted.

The Methods tab also helps define Additional Methods.

Business Logic, such as function modules, BAPIs, or methods in helper classes, can be accessed from the methods of all controllers.

8.19 COMPONENT / CUSTOM CONTROLLERS: SPECIAL ENTITIES

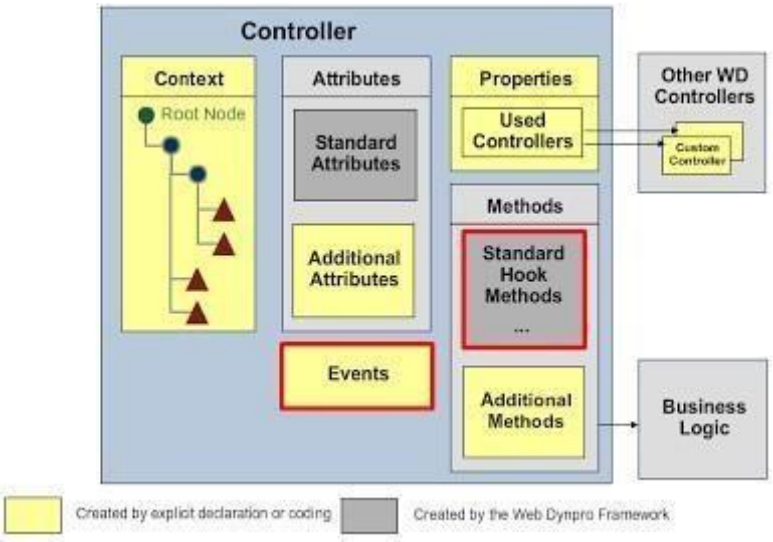


Fig. 8.17 Special Entities: Custom Controllers

You can create Events for component and custom controllers, and define them with the parameters of your choice.

You can register any method of any other controller, including view and window controllers, to the created events if the method is defined as an event handler one.

The component controller has three **additional hook methods**: WDDOBEFORENAVIGATION( ), WDDOPOSTPROCESSING( ), and WDDOAPPLICATIONSTATECHANGE( ).

8.20 VIEW CONTROLLERS: SPECIAL ENTITIES

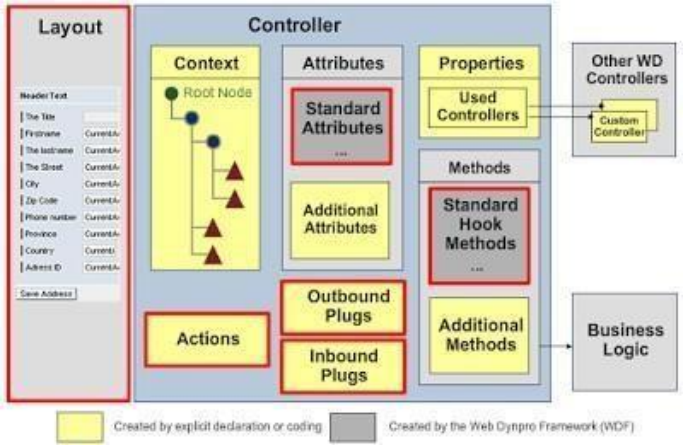


Fig. 8.18 Special Entities: View Controllers

An **action** links a client-side event, such as clicking a button in a browser, to an event handler method defined in the corresponding view controller. A navigation event is raised when an **outbound plug** is fired. An **inbound plug** is a navigation event handler that can be registered to a navigation request. **Layout** is used to create the interface between the user and the client.

8.21 WINDOW CONTROLLER ARCHITECTURE

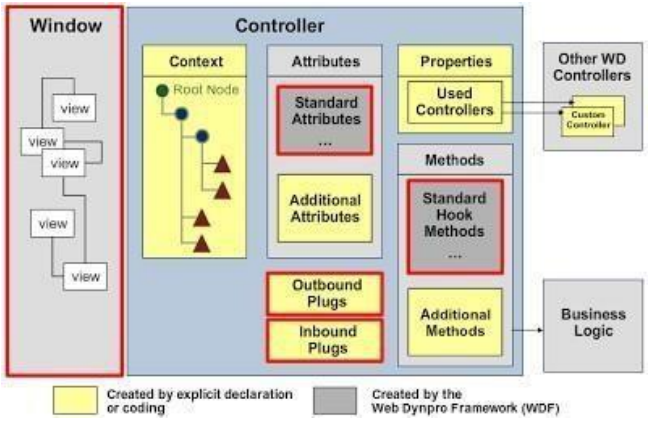


Fig. 8.19 Window Controller Architecture

Window controllers act as controllers without a UI (view layout). You embed all the views that are to be displayed when using a Web application in the window that is referred to by it. The Web Dynpro window embeds the views to be displayed, as well as the navigation links. Each Web Dynpro window contains outbound plugs and inbound plugs, just like views.

CHAPTER 9

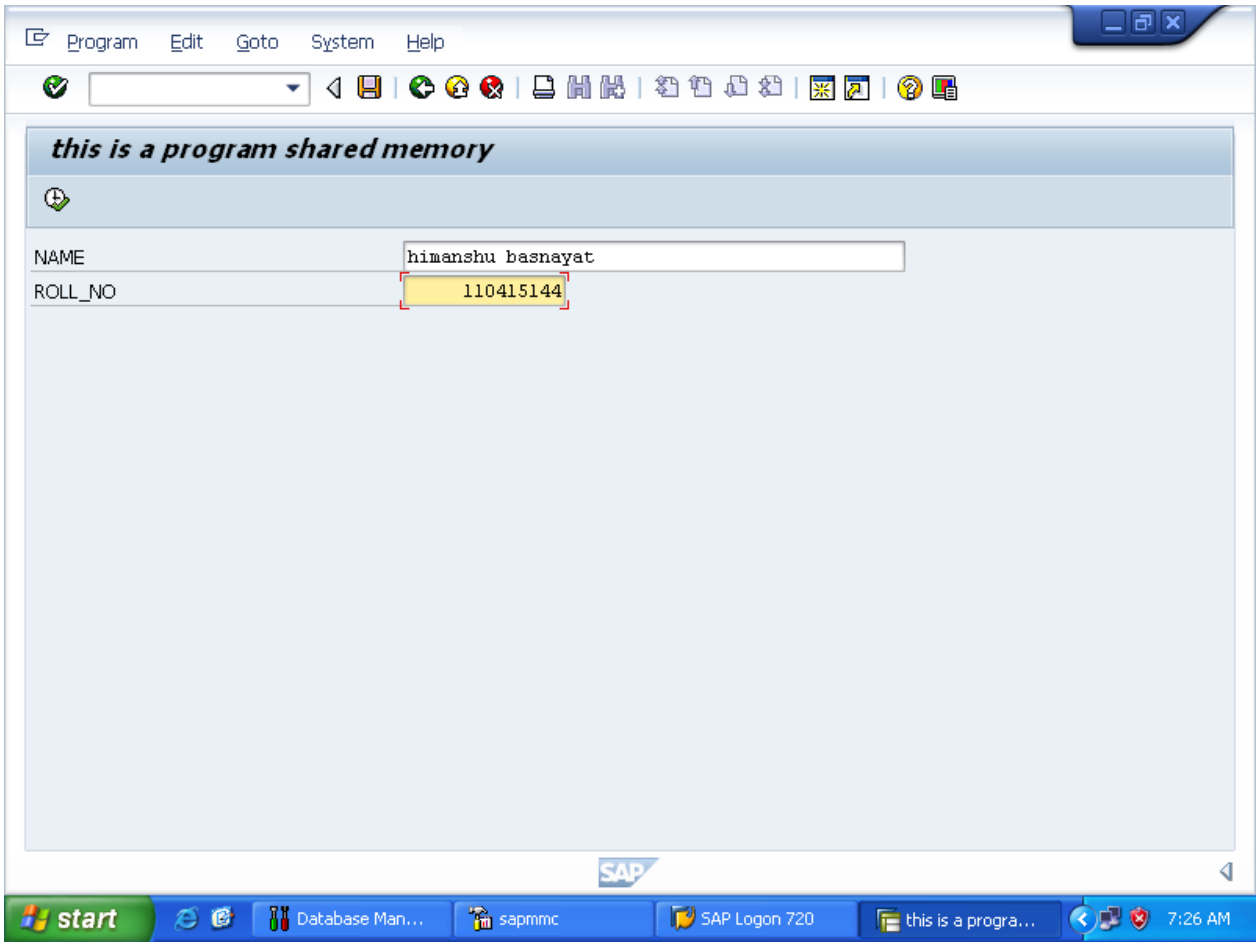
INTRODUCTION TO SAP

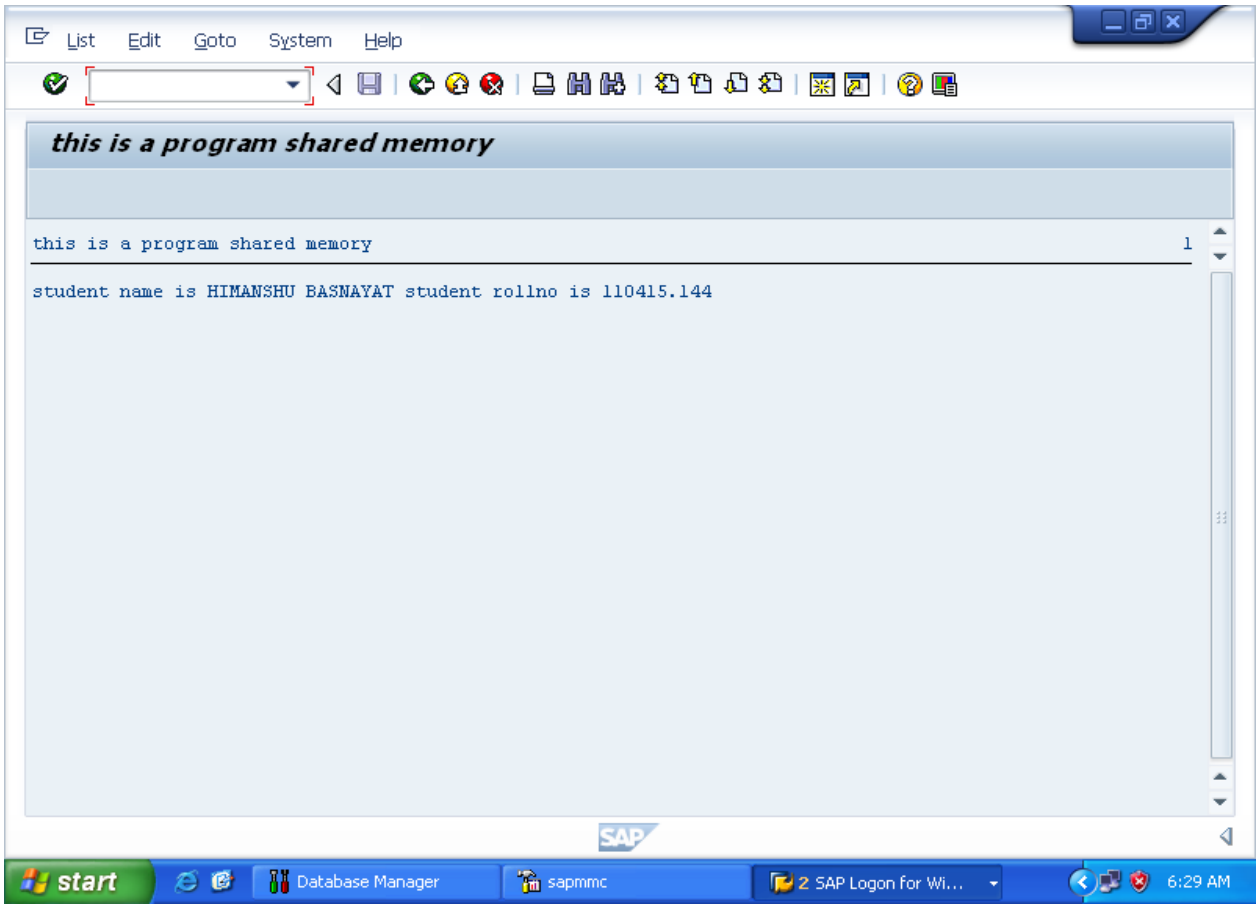
9.1 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS THE CONCEPT OF SHARED MEMORY.

```
*&---------------------------------------------------------------------*
*& Report Z_STUDENTS
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT Z_STUDENTS.
parameters: name TYPE ZNAME,
roll_no TYPE ZROLL_NO.
start-of-selection.
data: student_1 TYPE REF TO zstudents,
lr_handle TYPE REF TO zstudents_area.
try.
lr_handle = zstudents_area=>attach_for_write( ).
CREATE OBJECT student_1 AREA HANDLE lr_handle.
CALL METHOD student_1->get_attribute
exporting
iv_name = name
iv_roll_no = roll_no.
CALL METHOD student_1->display.
```

```
lr_handle->set_root( student_1 ).

lr_handle->detach_commit( ).

catch cx_shm_external_type.

endtry.
```

OUTPUT





9.2 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS ALV GRID CONTROL.

&---

*& Report ZALV

*&

&---

*&

*&

&---

REPORT ZALV.

DATA: it_spfli TYPE TABLE OF SPFLI.

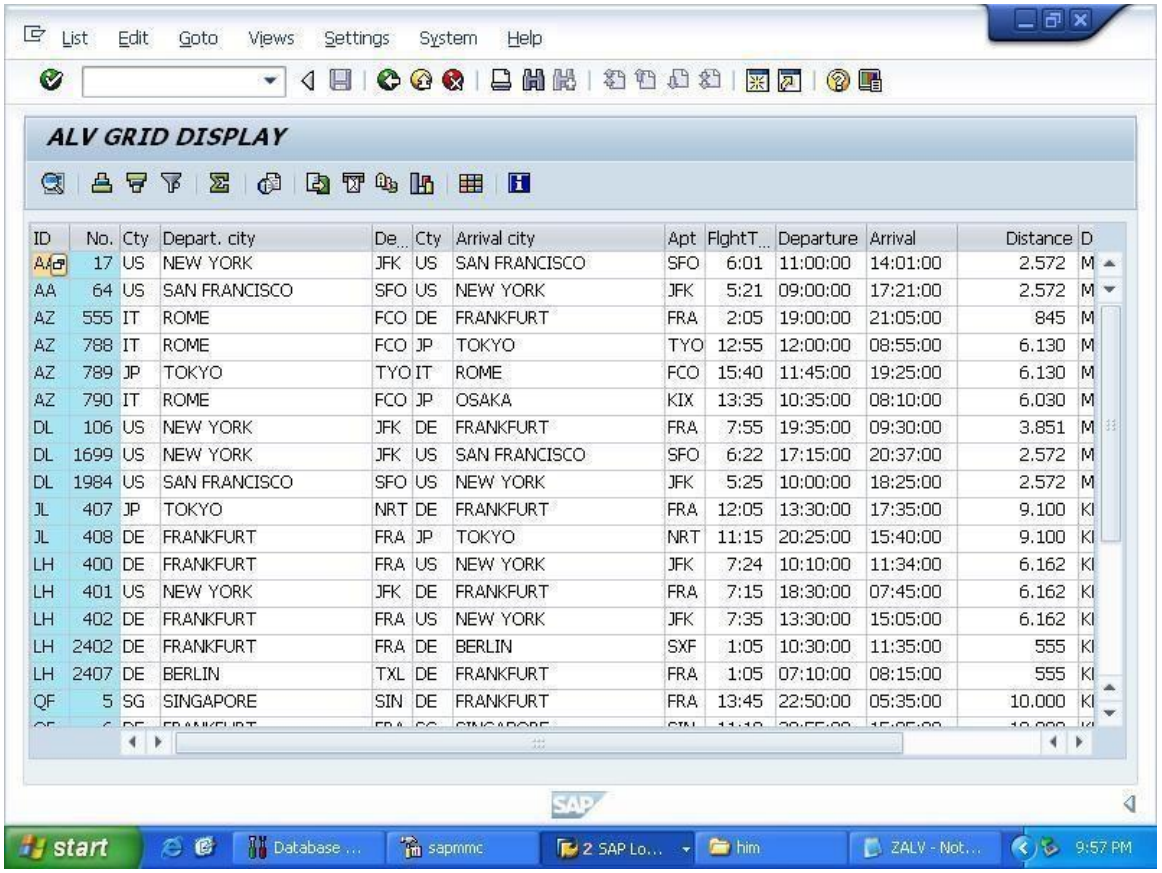
*SELECT * FROM SPFLI INTO TABLE it_spfli.*

CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'

```
EXPORTING
    * I_INTERFACE_CHECK = ' '
    * I_BYPASSING_BUFFER = ' '
    * I_BUFFER_ACTIVE = ' '
    * I_CALLBACK_PROGRAM = ' '
    * I_CALLBACK_PF_STATUS_SET = ' '
    * I_CALLBACK_USER_COMMAND = ' '
    * I_CALLBACK_TOP_OF_PAGE = ' '
    * I_CALLBACK_HTML_TOP_OF_PAGE = ' '
    * I_CALLBACK_HTML_END_OF_LIST = ' '
    I_STRUCTURE_NAME = 'SPFLI'
    * I_BACKGROUND_ID = ' '
    * I_GRID_TITLE =
    * I_GRID_SETTINGS =
    * IS_LAYOUT =
    * IT_FIELDCAT =
    * IT_EXCLUDING =
    * IT_SPECIAL_GROUPS =
    * IT_SORT =
    * IT_FILTER =
    * IS_SEL_HIDE =
    * I_DEFAULT = 'X'
    * I_SAVE = ' '
    * IS_VARIANT =
    * IT_EVENTS =
    * IT_EVENT_EXIT =
```

```
* IS_PRINT =
* IS_REPREP_ID =
* I_SCREEN_START_COLUMN = 0
* I_SCREEN_START_LINE = 0
* I_SCREEN_END_COLUMN = 0
* I_SCREEN_END_LINE = 0
* I_HTML_HEIGHT_TOP = 0
* I_HTML_HEIGHT_END = 0
* IT_ALV_GRAPHICS =
* IT_HYPERLINK =
* IT_ADD_FIELDCAT =
* IT_EXCEPT_QINFO =
* IR_SALV_FULLSCREEN_ADAPTER =
* IMPORTING
* E_EXIT_CAUSED_BY_CALLER =
* ES_EXIT_CAUSED_BY_USER =
TABLES
T_OUTTAB = it_spfli.
* EXCEPTIONS
* PROGRAM_ERROR = 1
* OTHERS = 2
.
IF SY-SUBRC <> 0.
* Implement suitable error handling here
ENDIF.
```

OUTPUT



ID	No.	Cty	Depart. city	De...	Cty	Arrival city	Apt	FlightT...	Departure	Arrival	Distance	D
AA	17	US	NEW YORK	JFK	US	SAN FRANCISCO	SFO	6:01	11:00:00	14:01:00	2.572	M
AA	64	US	SAN FRANCISCO	SFO	US	NEW YORK	JFK	5:21	09:00:00	17:21:00	2.572	M
AZ	555	IT	ROME	FCO	DE	FRANKFURT	FRA	2:05	19:00:00	21:05:00	845	M
AZ	788	IT	ROME	FCO	JP	TOKYO	TYO	12:55	12:00:00	08:55:00	6.130	M
AZ	789	JP	TOKYO	TYO	IT	ROME	FCO	15:40	11:45:00	19:25:00	6.130	M
AZ	790	IT	ROME	FCO	JP	OSAKA	KIX	13:35	10:35:00	08:10:00	6.030	M
DL	106	US	NEW YORK	JFK	DE	FRANKFURT	FRA	7:55	19:35:00	09:30:00	3.851	M
DL	1699	US	NEW YORK	JFK	US	SAN FRANCISCO	SFO	6:22	17:15:00	20:37:00	2.572	M
DL	1984	US	SAN FRANCISCO	SFO	US	NEW YORK	JFK	5:25	10:00:00	18:25:00	2.572	M
JL	407	JP	TOKYO	NRT	DE	FRANKFURT	FRA	12:05	13:30:00	17:35:00	9.100	KI
JL	408	DE	FRANKFURT	FRA	JP	TOKYO	NRT	11:15	20:25:00	15:40:00	9.100	KI
LH	400	DE	FRANKFURT	FRA	US	NEW YORK	JFK	7:24	10:10:00	11:34:00	6.162	KI
LH	401	US	NEW YORK	JFK	DE	FRANKFURT	FRA	7:15	18:30:00	07:45:00	6.162	KI
LH	402	DE	FRANKFURT	FRA	US	NEW YORK	JFK	7:35	13:30:00	15:05:00	6.162	KI
LH	2402	DE	FRANKFURT	FRA	DE	BERLIN	SXF	1:05	10:30:00	11:35:00	555	KI
LH	2407	DE	BERLIN	TXL	DE	FRANKFURT	FRA	1:05	07:10:00	08:15:00	555	KI
QF	5	SG	SINGAPORE	SIN	DE	FRANKFURT	FRA	13:45	22:50:00	05:35:00	10.000	KI

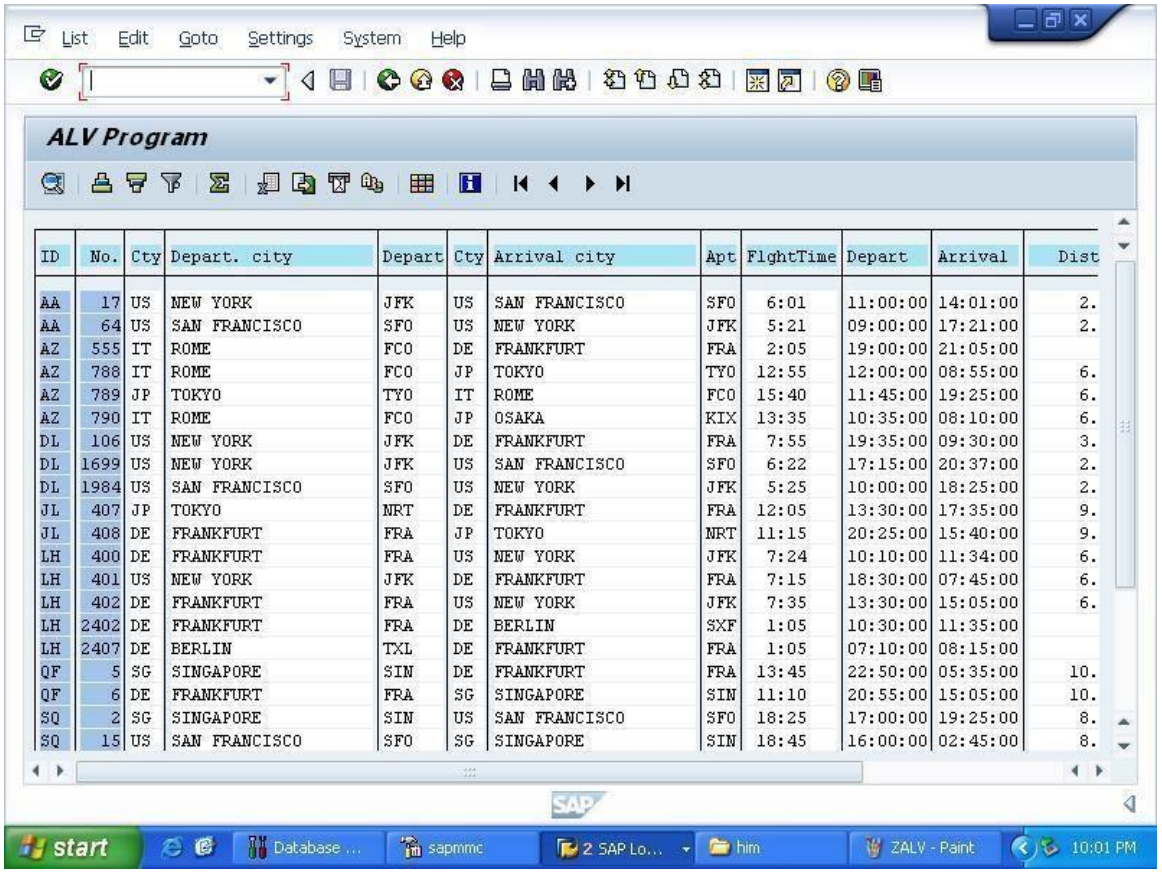
9.3 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS ALV GRID CONTROL LIST.

```
*&---------------------------------------------------------------------*
*& Report ZALV1
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZALV1.
DATA: it_spfli TYPE TABLE OF SPFLI.
```

```
SELECT * FROM SPFLI INTO TABLE it_spfli.
CALL FUNCTION 'REUSE_ALV_LIST_DISPLAY'
EXPORTING
* I_INTERFACE_CHECK = ' '
* I_BYPASSING_BUFFER =
* I_BUFFER_ACTIVE = ' '
* I_CALLBACK_PROGRAM = ' '
* I_CALLBACK_PF_STATUS_SET = ' '
* I_CALLBACK_USER_COMMAND = ' '
I_STRUCTURE_NAME = 'SPFLI'
* IS_LAYOUT =
* IT_FIELDCAT =
* IT_EXCLUDING =
* IT_SPECIAL_GROUPS =
* IT_SORT =
* IT_FILTER =
* IS_SEL_HIDE =
* I_DEFAULT = 'X'
* I_SAVE = ' '
* IS_VARIANT =
* IT_EVENTS =
* IT_EVENT_EXIT =
* IS_PRINT =
* IS_REPREP_ID =
* I_SCREEN_START_COLUMN = 0
* I_SCREEN_START_LINE = 0
```

```
* I_SCREEN_END_COLUMN = 0

* I_SCREEN_END_LINE = 0

* IR_SALV_LIST_ADAPTER =

* IT_EXCEPT_QINFO =

* I_SUPPRESS_EMPTY_DATA = ABAP_FALSE

* IMPORTING

* E_EXIT_CAUSED_BY_CALLER =

* ES_EXIT_CAUSED_BY_USER =

TABLES

T_OUTTAB = it_spfli.

* EXCEPTIONS

* PROGRAM_ERROR = 1

* OTHERS = 2

.

IF SY-SUBRC <> 0.

* Implement suitable error handling here

ENDIF.
```

OUTPUT



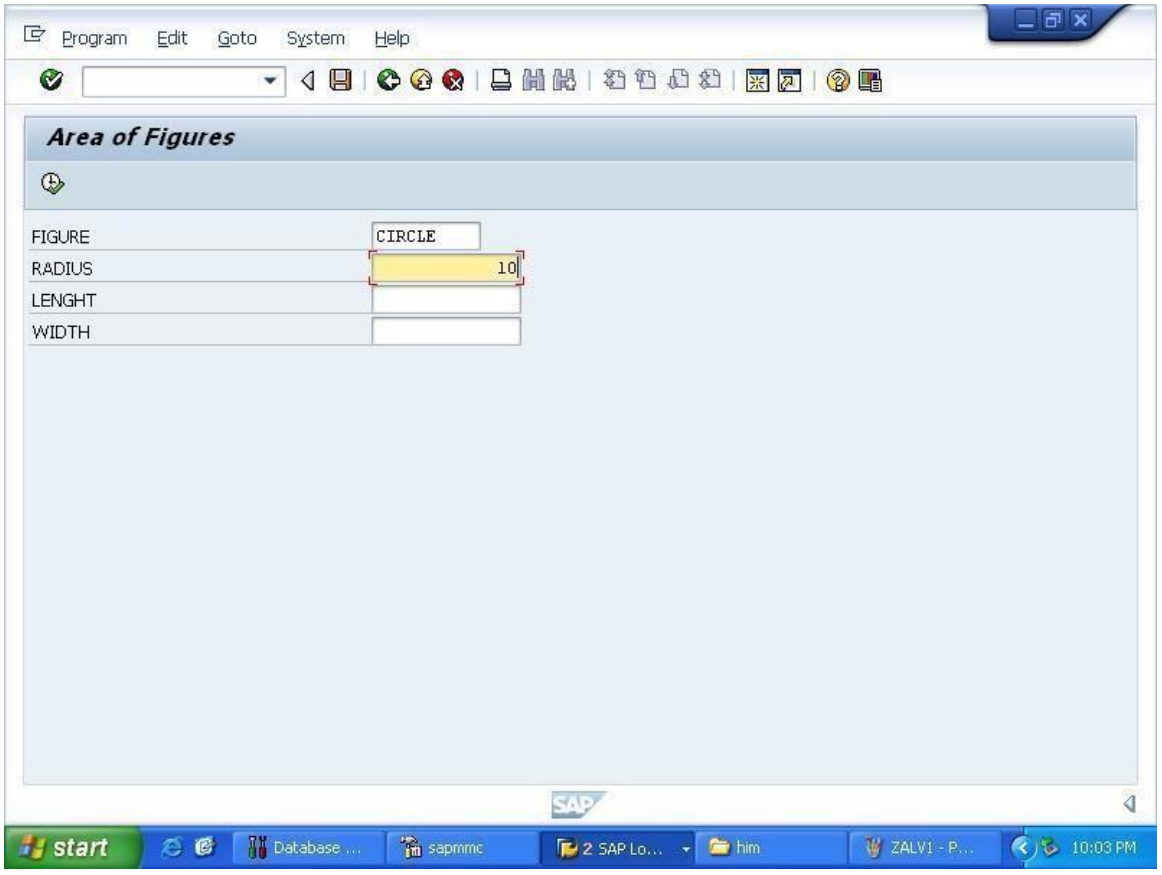
ID	No.	Cty	Depart. city	Depart	Cty	Arrival city	Apt	FlghtTime	Depart	Arrival	Dist
AA	17	US	NEW YORK	JFK	US	SAN FRANCISCO	SFO	6:01	11:00:00	14:01:00	2.
AA	64	US	SAN FRANCISCO	SFO	US	NEW YORK	JFK	5:21	09:00:00	17:21:00	2.
AZ	555	IT	ROME	FCO	DE	FRANKFURT	FRA	2:05	19:00:00	21:05:00	
AZ	788	IT	ROME	FCO	JP	TOKYO	TYO	12:55	12:00:00	08:55:00	6.
AZ	789	JP	TOKYO	TYO	IT	ROME	FCO	15:40	11:45:00	19:25:00	6.
AZ	790	IT	ROME	FCO	JP	OSAKA	KIX	13:35	10:35:00	08:10:00	6.
DL	106	US	NEW YORK	JFK	DE	FRANKFURT	FRA	7:55	19:35:00	09:30:00	3.
DL	1699	US	NEW YORK	JFK	US	SAN FRANCISCO	SFO	6:22	17:15:00	20:37:00	2.
DL	1984	US	SAN FRANCISCO	SFO	US	NEW YORK	JFK	5:25	10:00:00	18:25:00	2.
JL	407	JP	TOKYO	NRT	DE	FRANKFURT	FRA	12:05	13:30:00	17:35:00	9.
JL	408	DE	FRANKFURT	FRA	JP	TOKYO	NRT	11:15	20:25:00	15:40:00	9.
LH	400	DE	FRANKFURT	FRA	US	NEW YORK	JFK	7:24	10:10:00	11:34:00	6.
LH	401	US	NEW YORK	JFK	DE	FRANKFURT	FRA	7:15	18:30:00	07:45:00	6.
LH	402	DE	FRANKFURT	FRA	US	NEW YORK	JFK	7:35	13:30:00	15:05:00	6.
LH	2402	DE	FRANKFURT	FRA	DE	BERLIN	SXF	1:05	10:30:00	11:35:00	
LH	2407	DE	BERLIN	TXL	DE	FRANKFURT	FRA	1:05	07:10:00	08:15:00	
QF	5	SG	SINGAPORE	SIN	DE	FRANKFURT	FRA	13:45	22:50:00	05:35:00	10.
QF	6	DE	FRANKFURT	FRA	SG	SINGAPORE	SIN	11:10	20:55:00	15:05:00	10.
SQ	2	SG	SINGAPORE	SIN	US	SAN FRANCISCO	SFO	18:25	17:00:00	19:25:00	8.
SQ	15	US	SAN FRANCISCO	SFO	SG	SINGAPORE	SIN	18:45	16:00:00	02:45:00	8.

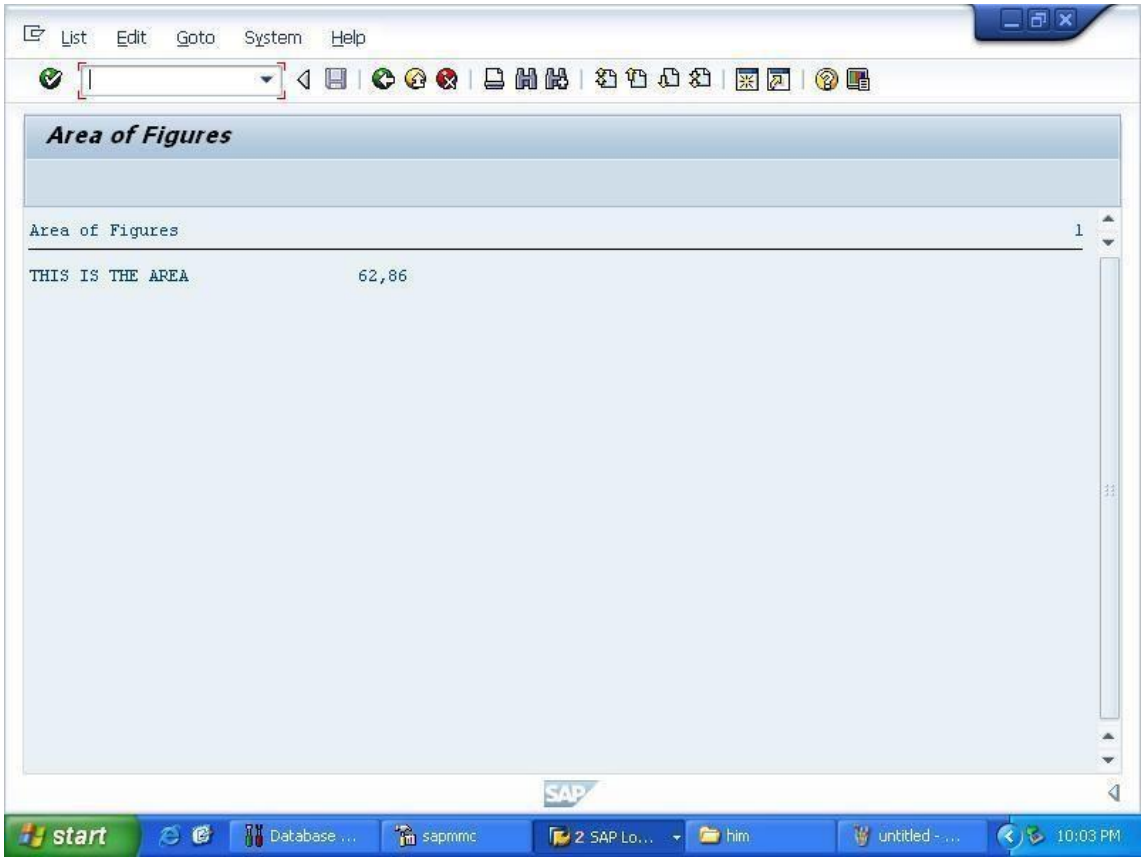
9.4 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS CONDITIONAL STRUCTURE.

```
*&---------------------------------------------------------------------*
*& Report ZAREA
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZAREA.
PARAMETERS: FIGURE TYPE C LENGTH 10 .
```

```
data: area type P LENGTH 10 DECIMALS 2.
if (
FIGURE = 'CIRCLE' or
FIGURE = 'RECTANGLE').
CASE FIGURE.
WHEN 'CIRCLE'.
PARAMETERs: RADIUS TYPE I.
AREA = 2 * 22 / 7 *  RADIUS.
WRITE: 'THIS IS THE AREA ',AREA.
WHEN 'RECTANGLE' .
PARAMETERs: LENGHT TYPE I,
WIDTH TYPE I.
AREA = LENGHT * WIDTH .
WRITE: 'THIS IS THE AREA ',AREA.
ENDCASE.
ENDIF.
```

OUTPUT

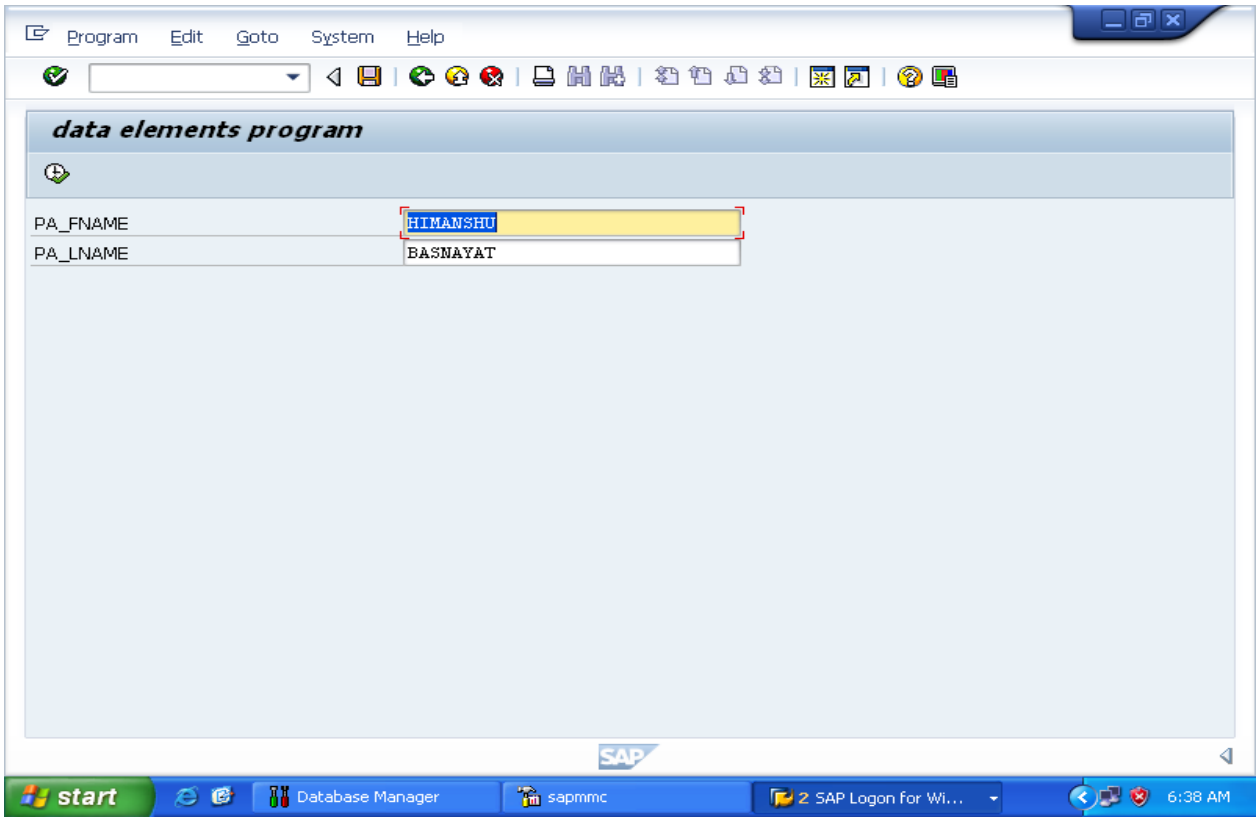

Program Edit Goto System Help
Area of Figures
FIGURE CIRCLE
RADIUS 10
LENGHT
WIDTH
start Database ... sapmmc 2 SAP Lo... him ZALV1 - P... 10:03 PM

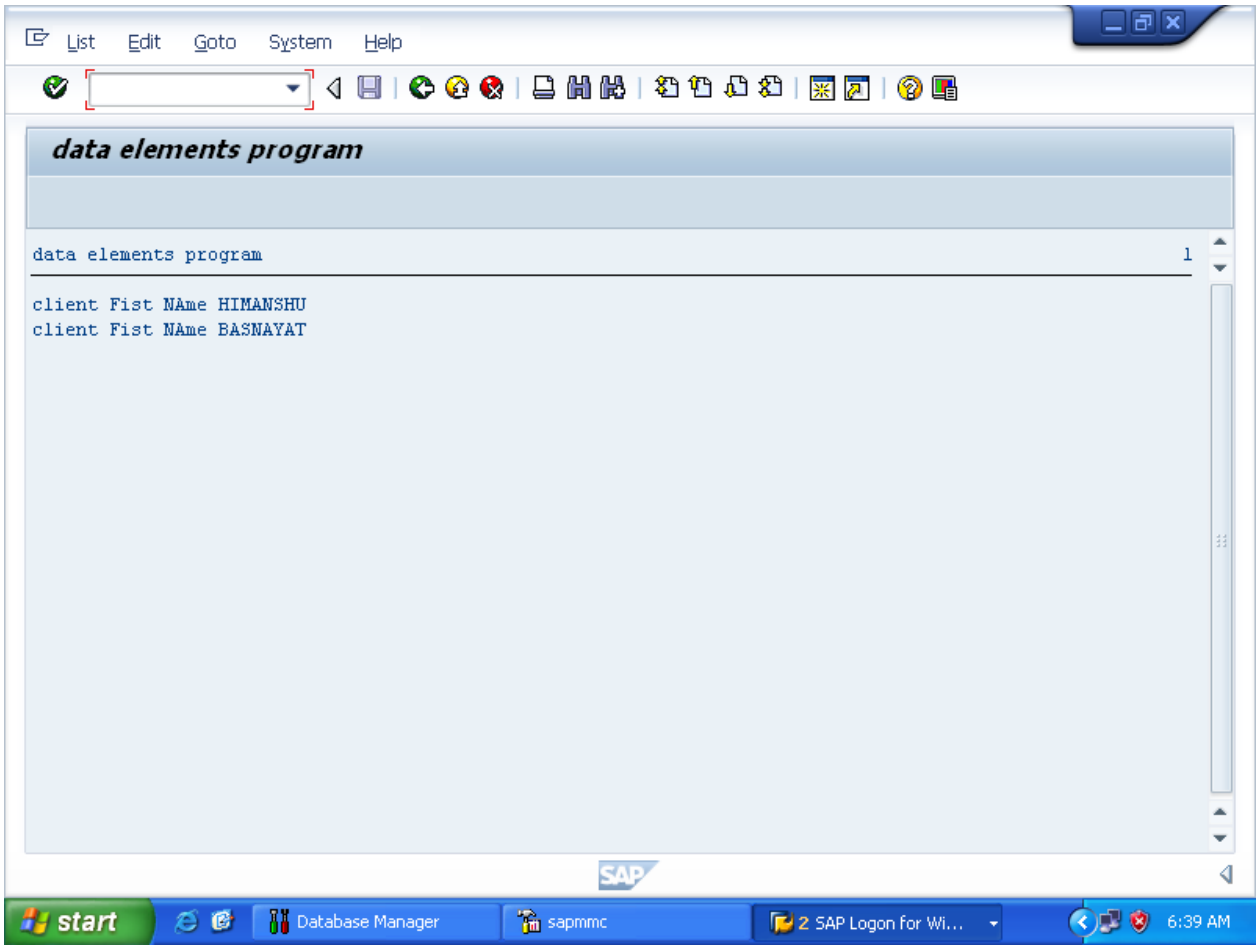

List Edit Goto System Help
Area of Figures
Area of Figures 1
THIS IS THE AREA 62,86
start Database ... sapmmc 2 SAP Lo... him untitled - ... 10:03 PM

9.5 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS DICTIONARY ELEMENTS SUCH AS DOMAIN AND DATA ELEMENT.

```
*&---------------------------------------------------------------------*
*& Report ZBC_DATA_ELEMENTS
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZBC_DATA_ELEMENTS.
PARAMETERS: pa_fname TYPE ZFIRSTNAME,
pa_lname TYPE ZLASTNAME.
NEW-LINE.
WRITE: /'Client First Name : ', pa_fname.
WRITE: /'Client Last Name : ', pa_lname.
NEW-LINE.
```

OUTPUT


Program Edit Goto System Help
data elements program
PA_FNAME
HIMANSHU
PA_LNAME
BASNAYAT
start
Database Manager
sapmmc
2 SAP Logon for Wi...
6:38 AM


List Edit Goto System Help
data elements program
data elements program 1
client Fist NAme HIMANSHU
client Fist NAme BASNAYAT
start
Database Manager
sapmmc
2 SAP Logon for Wi...
6:39 AM

9.6 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS UP-CASTING AND DOWN-CASTING IN INHERITANCE PROGRAM.

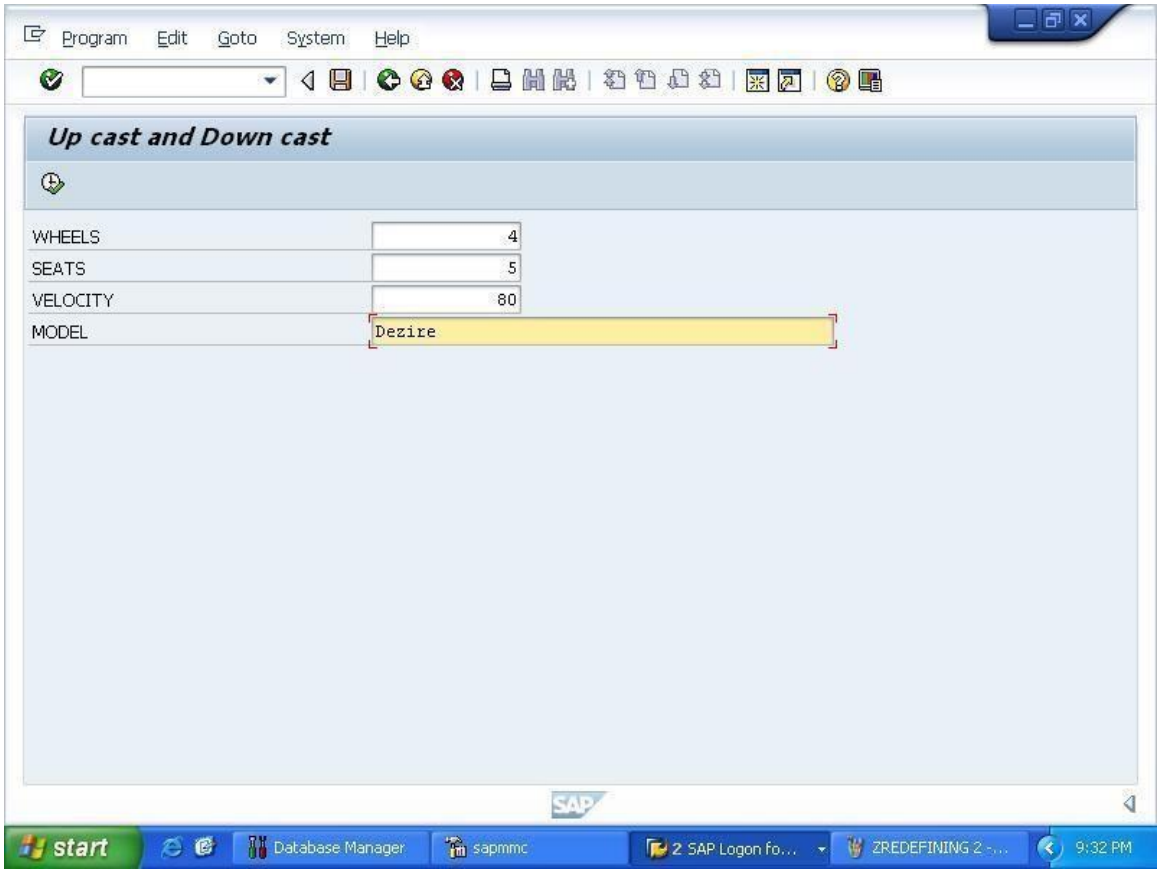
```
*&---------------------------------------------------------------------*
*& Report ZCASTING
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZCASTING.
CLASS vehicle DEFINITION.
PUBLIC SECTION.
METHODS: get_data IMPORTING iv_no_of_wheels TYPE I
iv_seats TYPE I
iv_velocity TYPE I, display_data.
PRIVATE SECTION.
DATA: mv_no_of_wheels TYPE I,
mv_seats TYPE I,
mv_velocity TYPE I.
ENDCLASS.
CLASS vehicle IMPLEMENTATION.
METHOD get_data.
mv_no_of_wheels = iv_no_of_wheels.
mv_seats = iv_seats.
mv_velocity = iv_velocity.
```

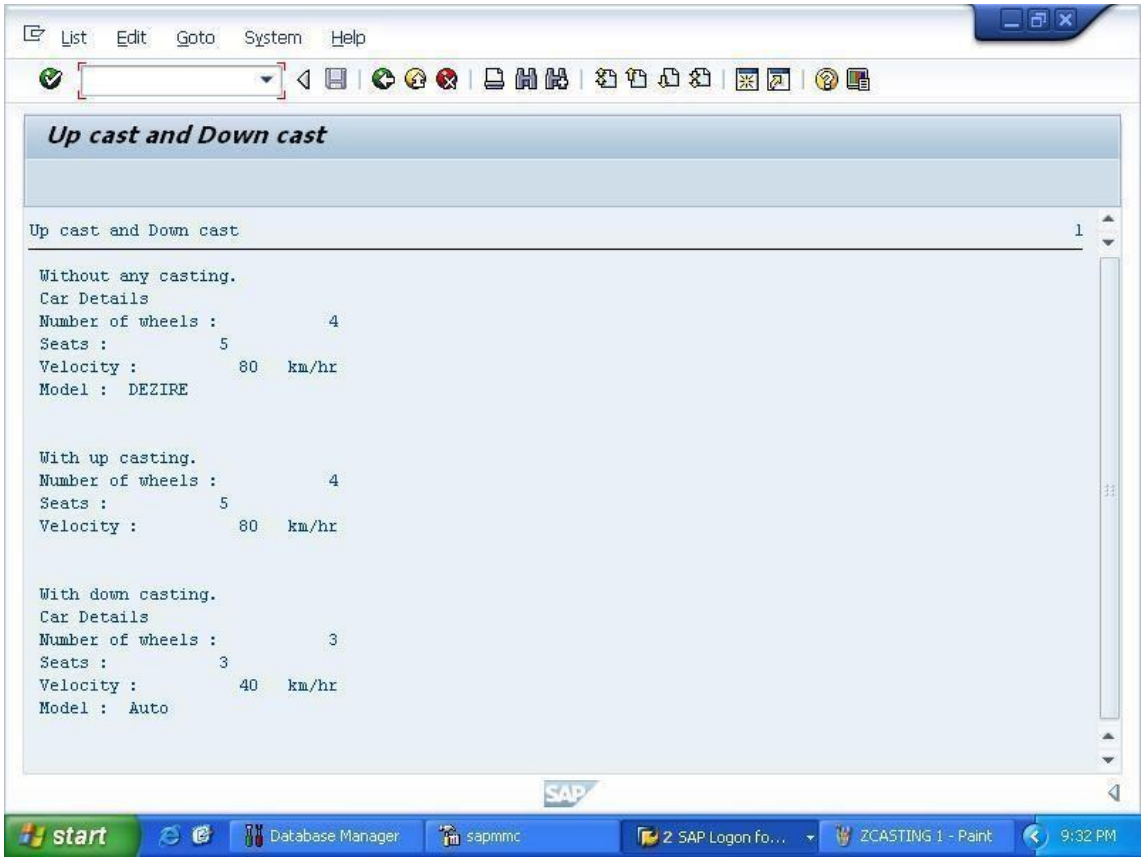
```
ENDMETHOD.
METHOD display_data.
WRITE: / ' Number of wheels : ', mv_no_of_wheels,
/ ' Seats : ', mv_seats,
/ ' Velocity : ', mv_velocity, ' km/hr'.
ENDMETHOD.
ENDCLASS.
CLASS car DEFINITION INHERITING FROM vehicle.
PUBLIC SECTION.
METHODS: get_car_data IMPORTING no_of_wheels TYPE I
seats TYPE I
velocity TYPE I
iv_model TYPE String, display.
PRIVATE SECTION.
DATA: mv_model TYPE String.
ENDCLASS.
CLASS car IMPLEMENTATION.
METHOD get_car_data.
CALL METHOD get_data
EXPORTING
iv_no_of_wheels = no_of_wheels
iv_seats = seats
iv_velocity = velocity.
mv_model = iv_model.
ENDMETHOD.
METHOD display.
```

```
WRITE: / ' Car Details  '.
CALL METHOD display_data.
WRITE: / ' Model : ', mv_model.
ENDMETHOD.
ENDCLASS.
START-OF-SELECTION.
PARAMETERS: wheels TYPE I,
seats TYPE I,
velocity TYPE I,
model TYPE String.
DATA: gv_vehicle TYPE REF TO vehicle,
gv_car TYPE REF TO car,
gv_car2 TYPE REF TO car.
CREATE OBJECT gv_car.
WRITE: / ' Without any casting. '.
CALL METHOD gv_car->get_car_data
EXPORTING
no_of_wheels = wheels
seats = seats
velocity = velocity
iv_model = model.
CALL METHOD gv_car->display.
SKIP 2.
WRITE: / ' With up casting. '.
gv_vehicle = gv_car.
CALL METHOD gv_vehicle->get_data
```

```
EXPORTING
iv_no_of_wheels = wheels
iv_seats = seats
iv_velocity = velocity.
CALL METHOD gv_vehicle->display_data.
SKIP 2.
WRITE: / ' With down casting. '.
gv_car2 ?= gv_vehicle.
CALL METHOD gv_car2->get_car_data
EXPORTING
no_of_wheels = 3
seats = 3
velocity = 40
iv_model = 'Auto'.
CALL METHOD gv_car2->display.
```

OUTPUT


Program Edit Goto System Help
Up cast and Down cast
WHEELS 4
SEATS 5
VELOCITY 80
MODEL Dezire
start
Database Manager
sapmmc
2 SAP Logon fo...
ZREDEFINING 2 -...
9:32 PM

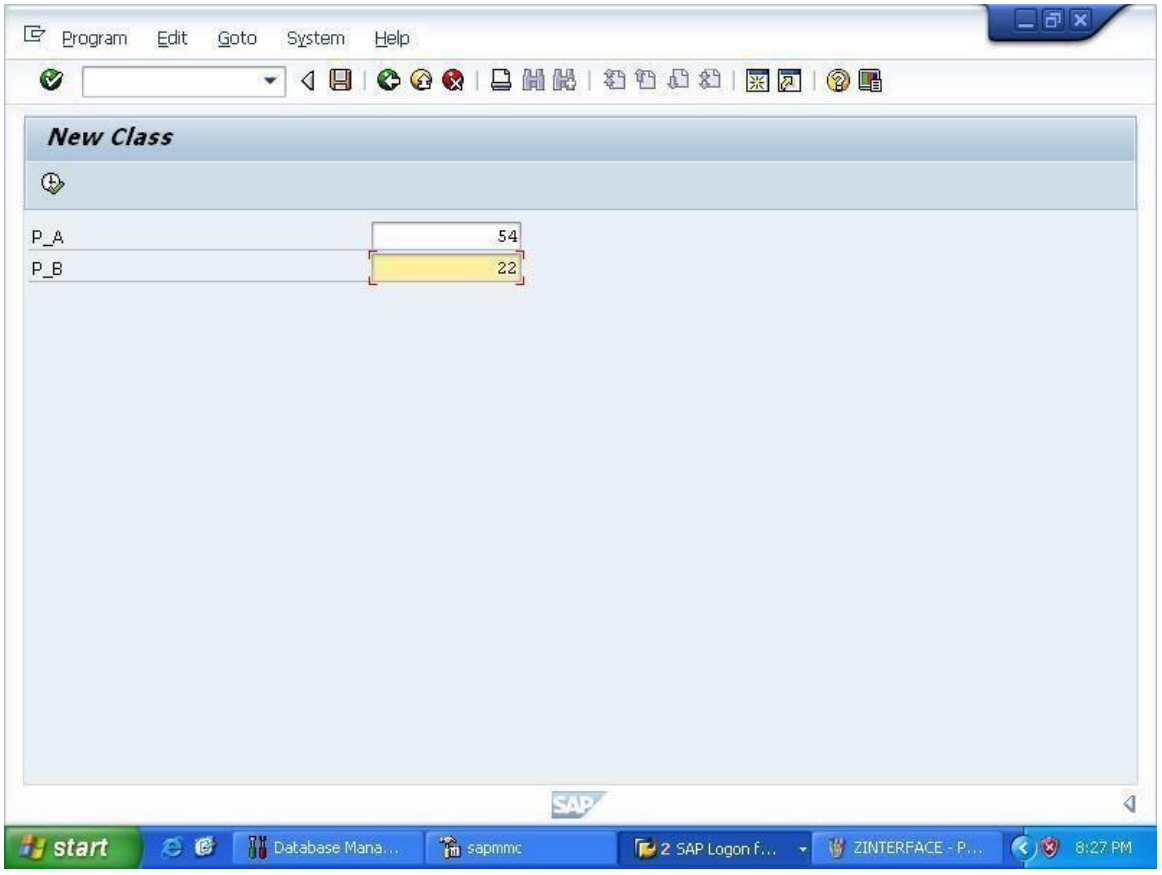

List Edit Goto System Help
Up cast and Down cast
Up cast and Down cast 1
Without any casting.
Car Details
Number of wheels : 4
Seats : 5
Velocity : 80 km/hr
Model : DEZIRE
With up casting.
Number of wheels : 4
Seats : 5
Velocity : 80 km/hr
With down casting.
Car Details
Number of wheels : 3
Seats : 3
Velocity : 40 km/hr
Model : Auto
start
Database Manager
sapmmc
2 SAP Logon fo...
ZCASTING 1 - Paint
9:32 PM

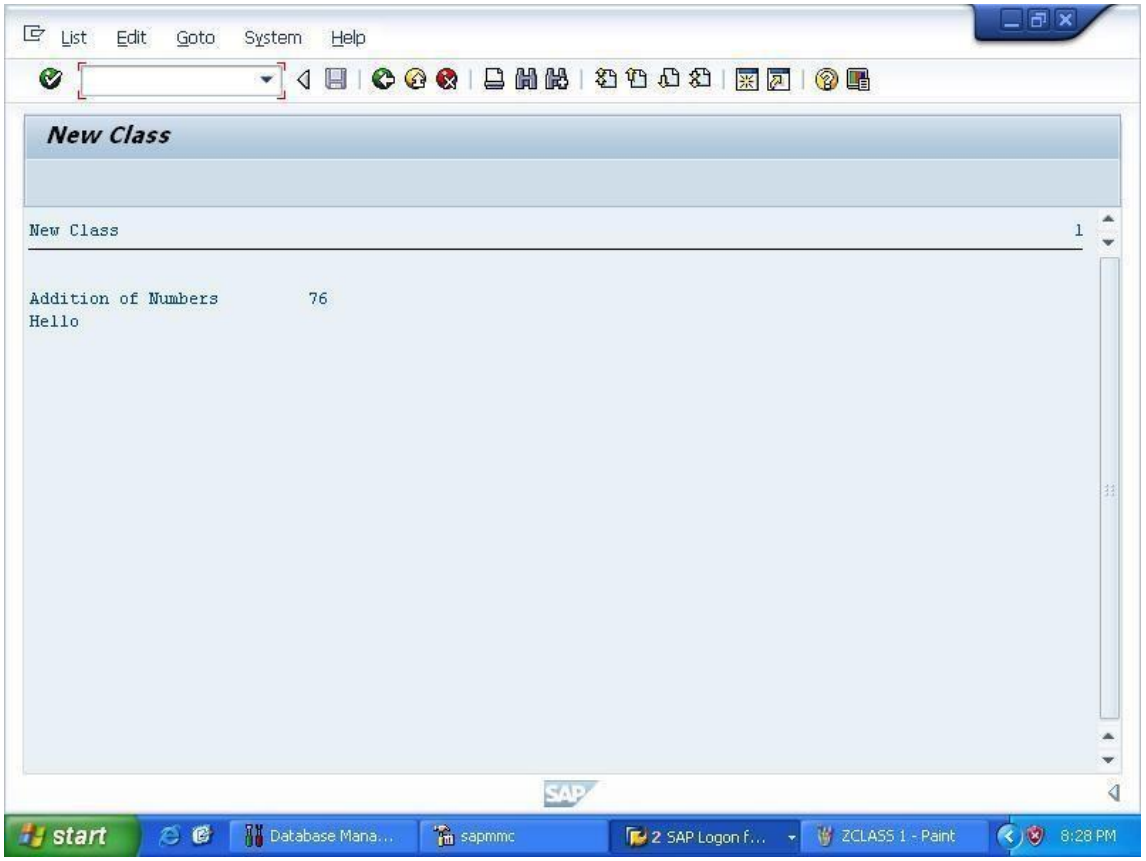
9.7 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS GLOBAL CLASSES.

```
*&---------------------------------------------------------------------*
*& Report ZCLASS
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZCLASS.
CLASS super_class DEFINITION.
PUBLIC SECTION.
METHODS: Addition1 IMPORTING g_a TYPE I
g_b TYPE I
EXPORTING g_c TYPE I.
ENDCLASS.
CLASS super_class IMPLEMENTATION.
METHOD Addition1.
g_c = g_a +  g_b.
ENDMETHOD.
ENDCLASS.
CLASS sub_class DEFINITION INHERITING FROM super_class.
PUBLIC SECTION.
METHODS: Display.
ENDCLASS.
CLASS sub_class IMPLEMENTATION.
```

```
METHOD Display.
WRITE:/ 'Hello'.
ENDMETHOD.
ENDCLASS.
START-OF-SELECTION.
PARAMETERS: p_a TYPE I,
p_b TYPE I.
DATA: Addition TYPE I.
DATA: Ref1 TYPE REF TO sub_class.
CREATE OBJECT Ref1.
CALL METHOD
Ref1->Addition1
EXPORTING g_a = p_a
g_b = p_b
IMPORTING g_c = Addition.
SKIP.
WRITE: 'Addition of Numbers', Addition.
Ref1->Display( ).
```

OUTPUT


Program Edit Goto System Help
New Class
P_A 54
P_B 22
start Database Mana... sapmmc 2 SAP Logon f... ZINTERFACE - P... 8:27 PM


List Edit Goto System Help
New Class
New Class 1
Addition of Numbers 76
Hello
start Database Mana... sapmmc 2 SAP Logon f... ZCLASS 1 - Paint 8:28 PM

9.8 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS CONSTRUCTORS IN LOCAL CLASSES.

```
*&---------------------------------------------------------------------*
*& Report ZCLASSCONSTR
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZCLASSCONSTR.
CLASS Students DEFINITION.
PUBLIC SECTION.
METHODS constructor IMPORTING iv_name TYPE STRING
iv_roll_no TYPE I.
CLASS-METHODS: get_branch IMPORTING iv_branch TYPE STRING. "display_branch.
METHODS display_data.
PRIVATE SECTION.
DATA: name TYPE STRING,
roll_no TYPE I.
CLASS-DATA branch TYPE STRING.
ENDCLASS.
CLASS Students IMPLEMENTATION.
METHOD constructor.
name = iv_name.
roll_no = iv_roll_no.
```

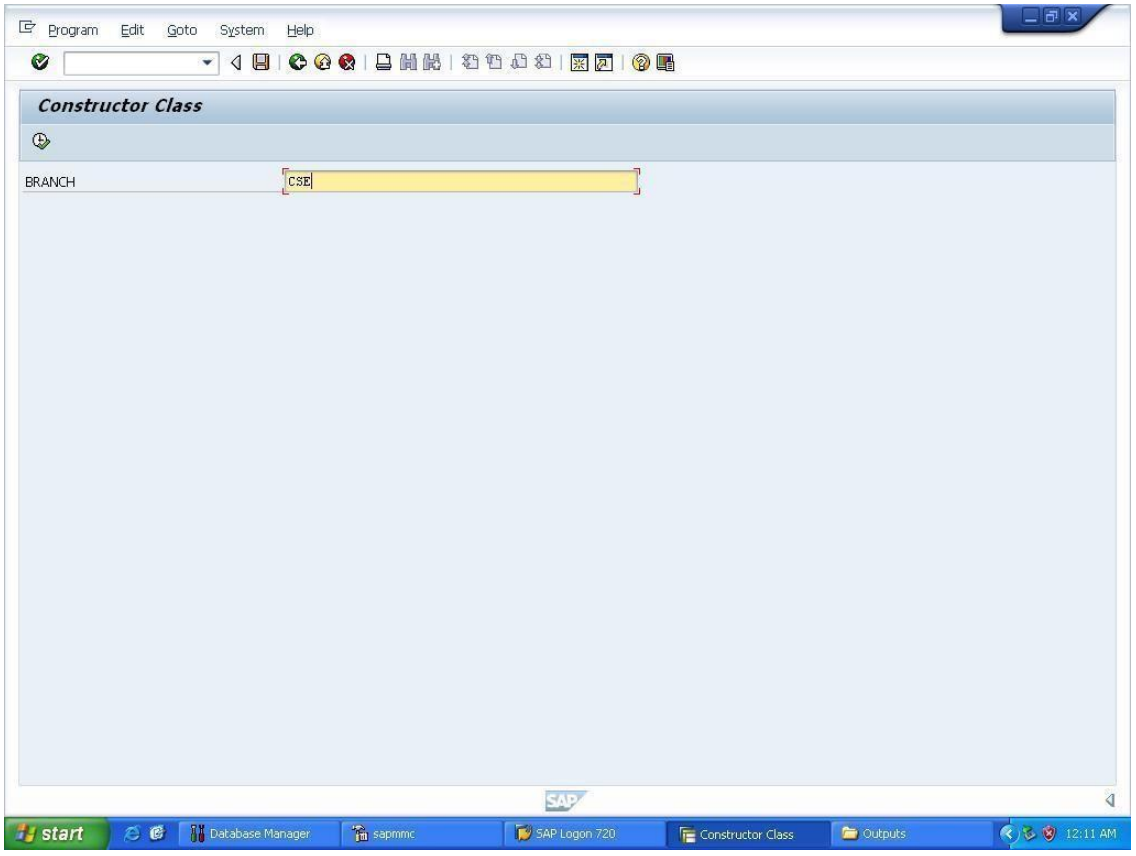
```
ENDMETHOD.
METHOD get_branch.
branch = iv_branch.
ENDMETHOD.
METHOD display_data.
WRITE: / ' Students Data ',
/ ' Name : ', name,
/ ' Roll No. : ', roll_no.
* ENDMETHOD.
* METHOD display_branch.
WRITE: / ' Branch : ', branch.
ENDMETHOD.
ENDCLASS.
DATA: go_student1 TYPE REF TO Students,
go_student2 LIKE go_student1.
START-OF-SELECTION.
CREATE OBJECT go_student1 EXPORTING iv_name = 'Priya'
iv_roll_no = 15000.
CREATE OBJECT go_student2 EXPORTING iv_name = 'Supriya'
iv_roll_no = 15056.
PARAMETERS: "name TYPE STRING,
"roll_no TYPE I,
branch TYPE STRING.
students=>get_branch( iv_branch = branch ).
go_student1->display_data( ).
* students=>display_branch( ).
```

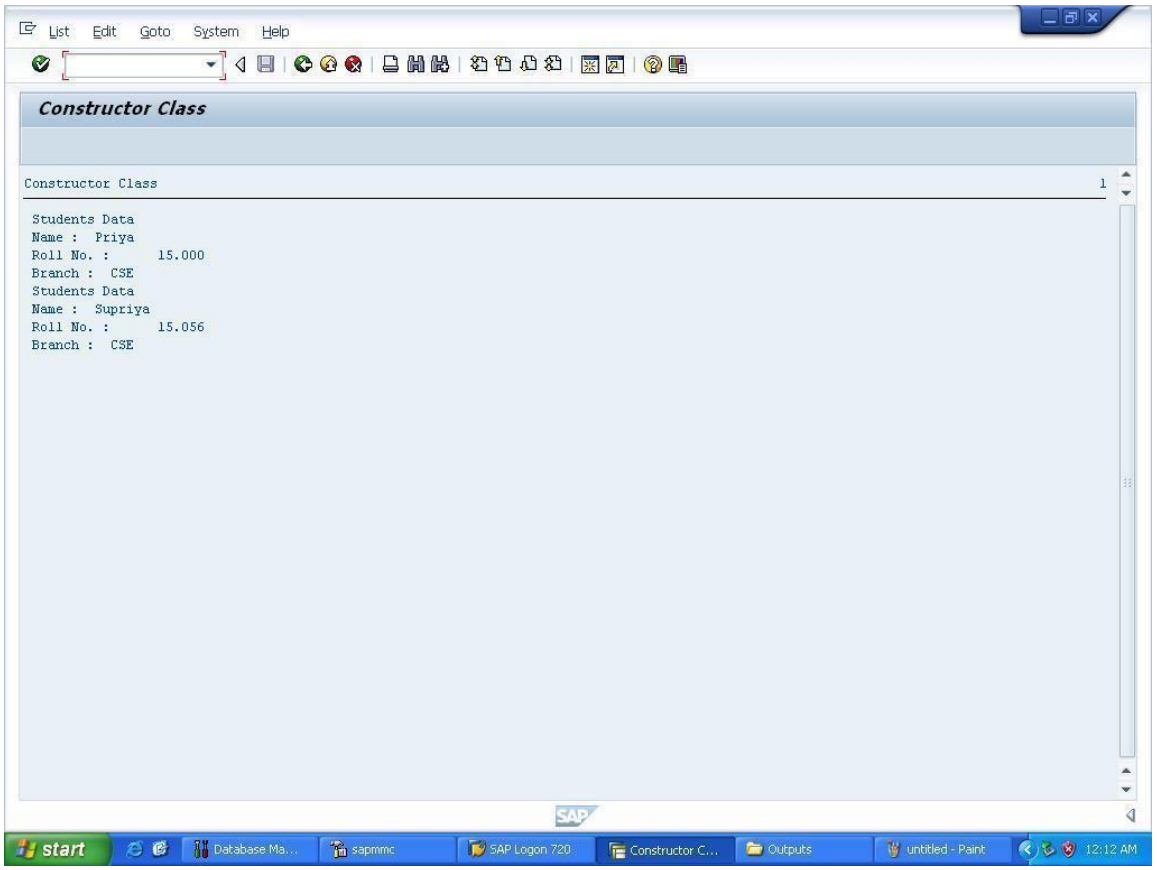
students=>get_branch(iv_branch = branch).

go_student2->display_data( ).

* students=>display_branch( ).

OUTPUT





9.9 WRITE AN EXECUTABLE PROGRAM THAT CREATES A LOCAL CLASS AND IMPLEMENTS STATIC ATTRIBUTES AND METHODS.

&---

*& Report ZCLASSEXAMPLE

*&

&---

*&

*&

&---

REPORT ZCLASSEXAMPLE.

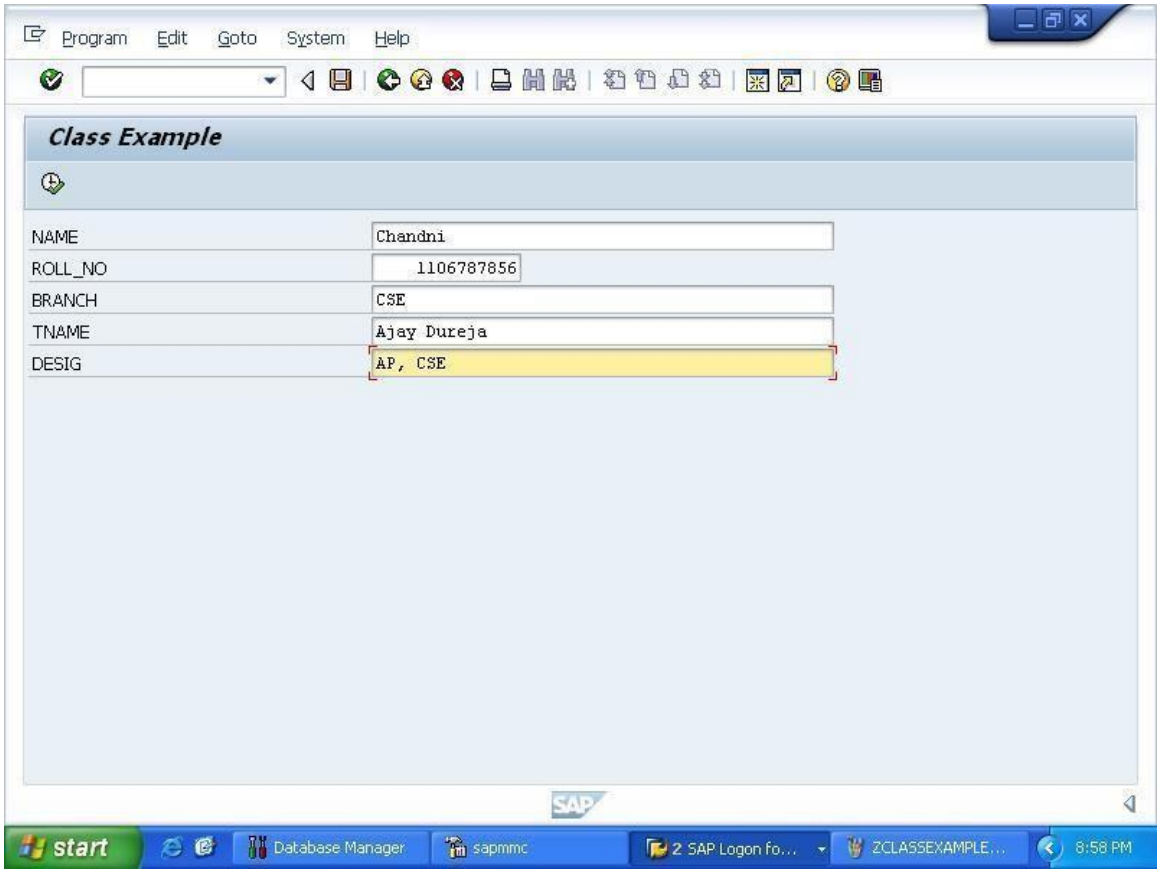
CLASS Students DEFINITION.

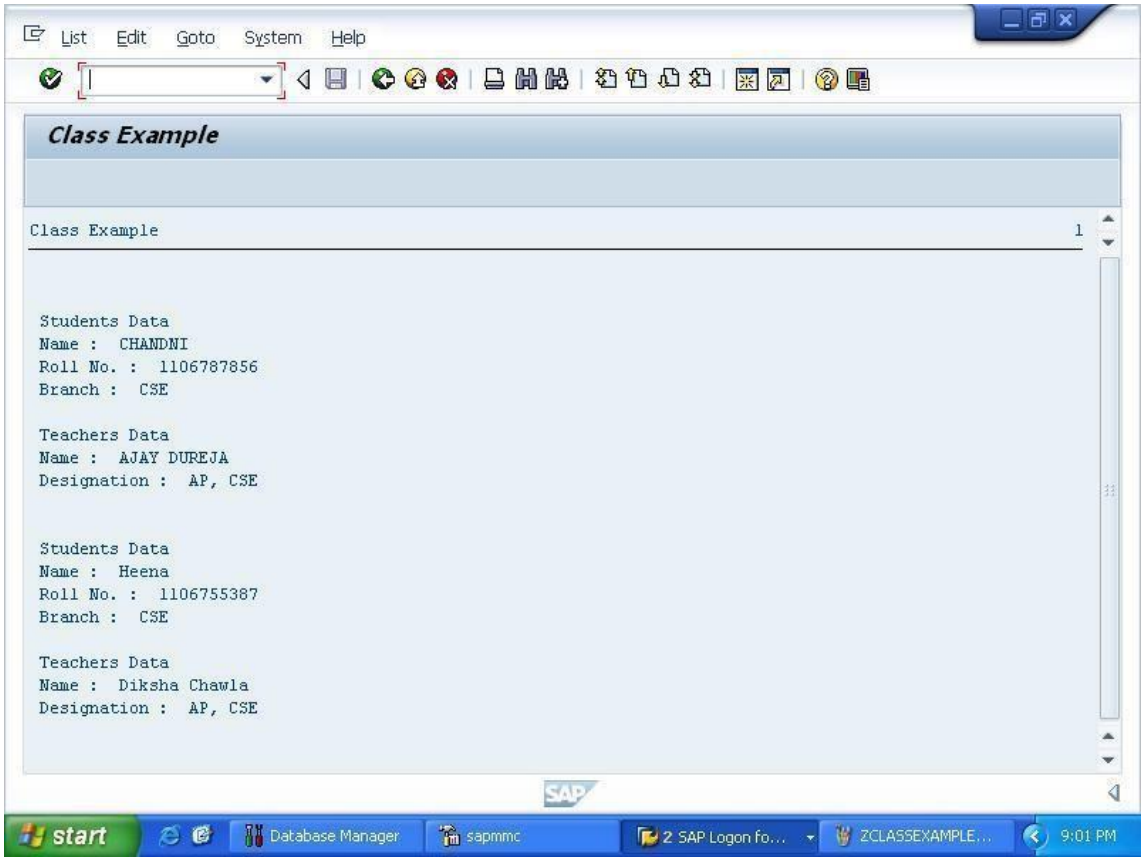
```
PUBLIC SECTION.
METHODS get_data IMPORTING iv_name TYPE STRING
iv_roll_no TYPE I.
CLASS-METHODS: get_branch IMPORTING iv_branch TYPE STRING. "display_branch.
METHODS display_data.
PRIVATE SECTION.
DATA: name TYPE STRING,
roll_no TYPE I.
CLASS-DATA branch TYPE STRING.
ENDCLASS.
CLASS Students IMPLEMENTATION.
METHOD get_data.
name = iv_name.
roll_no = iv_roll_no.
ENDMETHOD.
METHOD get_branch.
branch = iv_branch.
ENDMETHOD.
METHOD display_data.
SKIP 2.
WRITE: / ' Students Data ',
/ ' Name : ', name,
/ ' Roll No. : ', roll_no.
* ENDMETHOD.
* METHOD display_branch.
```

```
WRITE: / ' Branch : ', branch.
ENDMETHOD.
ENDCLASS.
CLASS teacher DEFINITION INHERITING FROM students.
PUBLIC SECTION.
METHODS: get_tdata IMPORTING iv_tname TYPE STRING
iv_desig TYPE STRING, display_tdata.
PRIVATE SECTION.
DATA: tname TYPE STRING,
desig TYPE STRING.
ENDCLASS.
CLASS teacher IMPLEMENTATION.
METHOD get_tdata.
tname = iv_tname.
desig = iv_desig.
ENDMETHOD.
METHOD display_tdata.
SKIP.
WRITE: / ' Teachers Data ',
/ ' Name : ', tname,
/ ' Designation : ', desig.
ENDMETHOD.
ENDCLASS.
DATA: go_student1 TYPE REF TO teacher,
go_student2 LIKE go_student1.
START-OF-SELECTION.
```

```
CREATE OBJECT go_student1.
CREATE OBJECT go_student2.
PARAMETERS: name TYPE STRING,
roll_no TYPE I,
branch  TYPE  STRING,
tname  TYPE  STRING,
desig TYPE STRING.
go_student1->get_data( iv_name = name
iv_roll_no = roll_no ).
go_student1->get_tdata( iv_tname = tname
iv_desig = desig ).
students=>get_branch( iv_branch = branch ).
go_student1->display_data( ).
go_student1->display_tdata( ).
* students=>display_branch( ).
go_student2->get_data( iv_name = 'Heena'
iv_roll_no = 1106755387 ).
go_student2->get_tdata( iv_tname = 'Diksha Chawla'
iv_desig = 'AP, CSE' ).
students=>get_branch( iv_branch = branch ).
go_student2->display_data( ).
go_student2->display_tdata( ).
* students=>display_branch( ).
```

OUTPUT


Program Edit Goto System Help
Class Example
NAME Chandni
ROLL_NO 1106787856
BRANCH CSE
TNAME Ajay Dureja
DESIG AP, CSE
start Database Manager sapmmc 2 SAP Logon fo... ZCLASSEXAMPLE... 8:58 PM


List Edit Goto System Help
Class Example
Class Example 1
Students Data
Name : CHANDNI
Roll No. : 1106787856
Branch : CSE
Teachers Data
Name : AJAY DUREJA
Designation : AP, CSE
Students Data
Name : Heena
Roll No. : 1106755387
Branch : CSE
Teachers Data
Name : Diksha Chawla
Designation : AP, CSE
start Database Manager sapmmc 2 SAP Logon fo... ZCLASSEXAMPLE... 9:01 PM

9.10 WRITE AN EXECUTABLE PROGRAM THAT READS THE CURRENT SYSTEM DATE AND WRITE IN YOUR OWN LANGUAGE IN TEXT FORMAT. EX. 20140727 SHOULD BE WRITTEN AS JULY THE TWENTY-SEVENTH, 2014

```
*&---------------------------------------------------------------------*
*& Report ZDATEPROCESS
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZDATEPROCESS.
DATA: DATE TYPE C LENGTH 100,
DD(2),
Y(2),
X(2),
MM(2),
YYYY(4).
YYYY = SY-DATUM+0(4).
MM = SY-DATUM+4(2).
DD = SY-DATUM+6(2).
CASE MM.
WHEN 01.
CONCATENATE DATE 'January the ' INTO DATE.
WHEN 02.
CONCATENATE DATE 'February the ' INTO DATE.
```

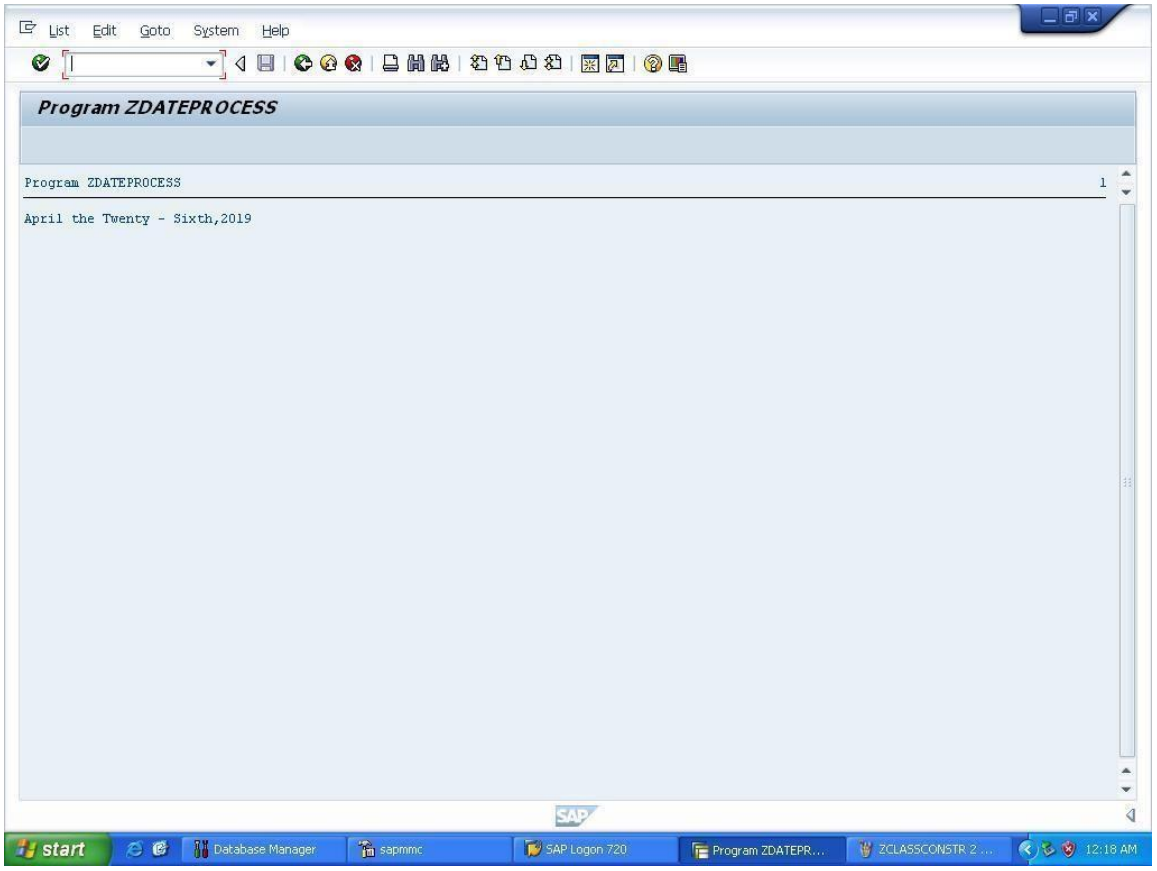
```
WHEN 03.
CONCATENATE DATE 'March the ' INTO DATE.
WHEN 04.
CONCATENATE DATE 'April the ' INTO DATE.
WHEN 05.
CONCATENATE DATE 'May the ' INTO DATE.
WHEN 06.
CONCATENATE DATE 'June the ' INTO DATE.
WHEN 07.
CONCATENATE DATE 'July the ' INTO DATE.
WHEN 08.
CONCATENATE DATE 'August the ' INTO DATE.
WHEN 09.
CONCATENATE DATE 'September the ' INTO DATE.
WHEN 10.
CONCATENATE DATE 'October the ' INTO DATE.
WHEN 11.
CONCATENATE DATE 'November the ' INTO DATE.
WHEN 12.
CONCATENATE DATE 'December the ' INTO DATE.
ENDCASE.
X = DD / 10.
Y = DD MOD 10.
CASE X .
WHEN 01.
CASE Y.
```

```
WHEN 01.
CONCATENATE DATE ' Eleventh, ' INTO DATE.
WHEN 02.
CONCATENATE DATE ' Twelfth, ' INTO DATE.
WHEN 03.
CONCATENATE DATE ' Thirteenth, ' INTO DATE.
WHEN 04.
CONCATENATE DATE ' Fourteenth, ' INTO DATE.
WHEN 05.
CONCATENATE DATE ' Fifteenth, ' INTO DATE.
WHEN 06.
CONCATENATE DATE ' Sixteenth, ' INTO DATE.
WHEN 07.
CONCATENATE DATE ' Seventeenth, ' INTO DATE.
WHEN 08.
CONCATENATE DATE ' Eighteenth, ' INTO DATE.
WHEN 09.
CONCATENATE DATE ' Nineteenth, ' INTO DATE.
WHEN 00.
CONCATENATE DATE ' Tenth, ' INTO DATE.
ENDCASE.
WHEN 02.
IF y = 0.
CONCATENATE DATE ' Twentieth, ' INTO DATE.
ELSE.
CONCATENATE DATE ' Twenty- ' INTO DATE.
```

```
CASE Y.
WHEN 01.
CONCATENATE DATE ' First, ' INTO DATE.
WHEN 02.
CONCATENATE DATE ' Second, ' INTO DATE.
WHEN 03.
CONCATENATE DATE ' Third, ' INTO DATE.
WHEN 04.
CONCATENATE DATE ' Fourth, ' INTO DATE.
WHEN 05.
CONCATENATE DATE ' Fifth, ' INTO DATE.
WHEN 06.
CONCATENATE DATE ' Sixth, ' INTO DATE.
WHEN 07.
CONCATENATE DATE ' Seventh, ' INTO DATE.
WHEN 08.
CONCATENATE DATE ' Eighth, ' INTO DATE.
WHEN 09.
CONCATENATE DATE ' Nineth, ' INTO DATE.
ENDCASE.
ENDIF.
WHEN 03.
IF y = 0.
CONCATENATE DATE ' Thirtieth,' INTO DATE.
ELSEIF y = 1.
CONCATENATE DATE ' Thirty-First, ' INTO DATE.
```

```
ENDIF.
WHEN 00.
CASE Y.
WHEN 01.
CONCATENATE DATE ' First, ' INTO DATE.
WHEN 02.
CONCATENATE DATE ' Second, ' INTO DATE.
WHEN 03.
CONCATENATE DATE ' Third, ' INTO DATE.
WHEN 04.
CONCATENATE DATE ' Fourth, ' INTO DATE.
WHEN 05.
CONCATENATE DATE ' Fifth, ' INTO DATE.
WHEN 06.
CONCATENATE DATE ' Sixth, ' INTO DATE.
WHEN 07.
CONCATENATE DATE ' Seventh, ' INTO DATE.
WHEN 08.
CONCATENATE DATE ' Eighth, ' INTO DATE.
WHEN 09.
CONCATENATE DATE ' Nineth, ' INTO DATE.
ENDCASE.
ENDCASE.
" CONCATENATE DATE ' ' INTO DATE.
CONCATENATE DATE YYYY INTO DATE.
WRITE: DATE.
```

OUTPUT



9.11 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS EVENT HANDLING IN LOCAL CLASSES.

```
*&---------------------------------------------------------------------*
*& Report ZEVENTHANDLING
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZEVENTHANDLING.
CLASS cl_main DEFINITION.
```

```
PUBLIC SECTION.
DATA: num1 TYPE I.
METHODS pro IMPORTING num2 TYPE I.
EVENTS cutoff.
ENDCLASS.
CLASS cl_main IMPLEMENTATION.
METHOD pro.
num1 = num2.
IF num2 >= 2.
RAISE EVENT cutoff.
ELSE.
num1 = num1 + 20.
WRITE: / 'Number 1 Value is : ', num1.
ENDIF.
ENDMETHOD.
ENDCLASS.
CLASS cl_event_handler DEFINITION.
PUBLIC SECTION.
METHODS handling_cutoff FOR EVENT cutoff OF cl_main.
ENDCLASS.
CLASS cl_event_handler IMPLEMENTATION.
METHOD handling_cutoff.
WRITE: / 'Handling CUTOFF Event.',
/ 'Event has been processed.'.
ENDMETHOD.
ENDCLASS.
```

START-OF-SELECTION.

DATA: main1 TYPE REF TO cl_main,

eventhandler1 TYPE REF TO cl_event_handler.

CREATE OBJECT main1.

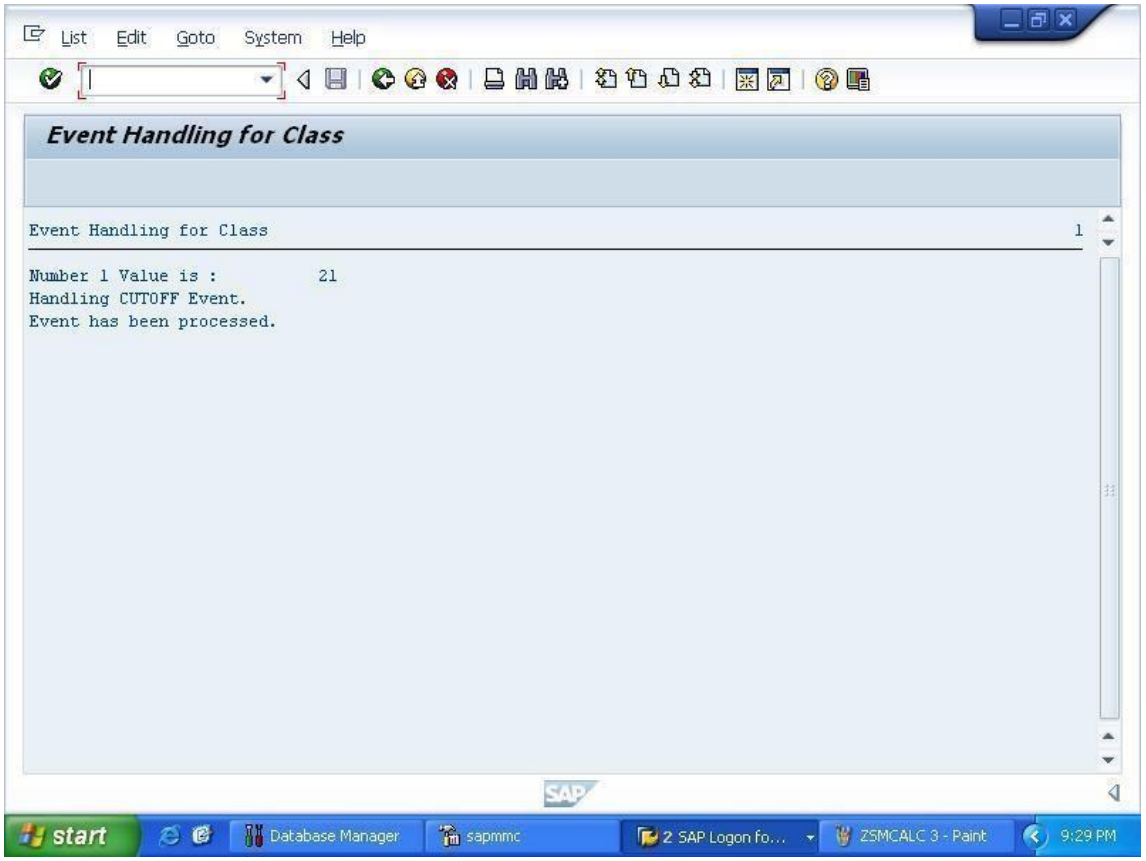
CREATE OBJECT eventhandler1.

SET HANDLER eventhandler1->handling_cutoff FOR main1.

main1->pro(1).

main1->pro(4).

OUTPUT



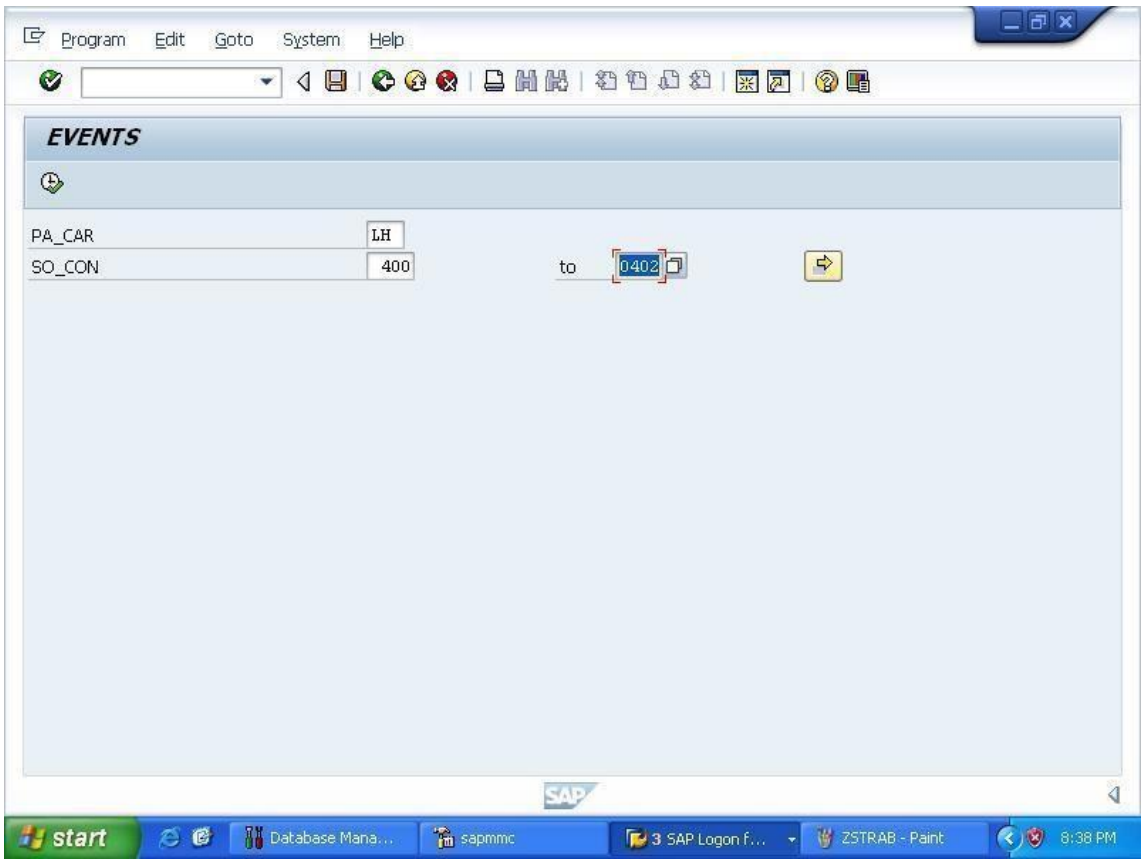
9.12 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS THE EVENTS AND AUTHORIZATION CHECK OF ABAP REPORT.

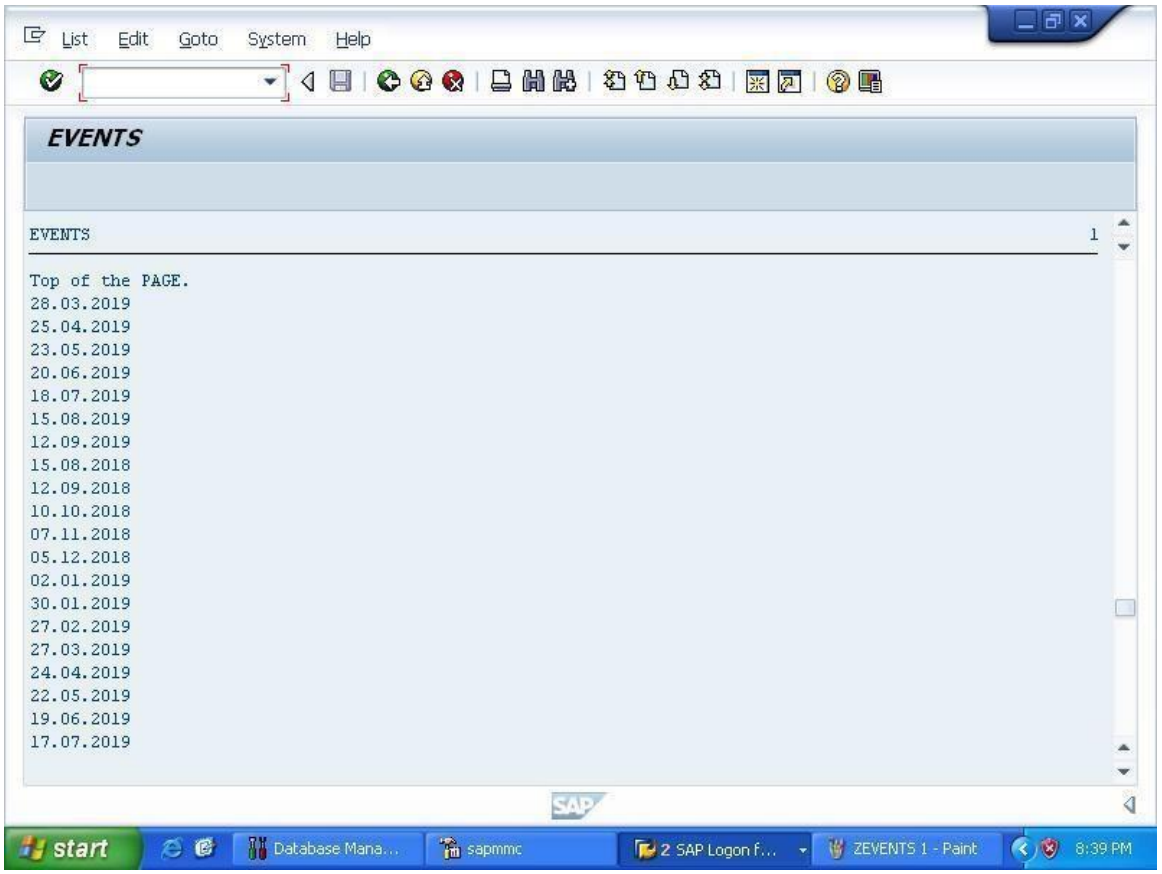
```
*&---------------------------------------------------------------------*
*& Report ZEVENTS
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZEVENTS.
TABLES SFLIGHT.
PARAMETERS: pa_car TYPE SFLIGHT-CARRID.
DATA: gs_sfli TYPE SFLIGHT.
SELECT-OPTIONS: so_con FOR SFLIGHT-CONNID.
CONSTANTS: gc_activity_display TYPE activ_auth VALUE '03'.
INITIALIZATION.
pa_car = 'LH'.
AT SELECTION-SCREEN.
CALL FUNCTION 'Z_AUTH_CHECK'
EXPORTING
IV_CARRID = pa_car
IV_ACTIVITY = gc_activity_display
* EXCEPTIONS
* NO_AUTHORITY = 1
* WRONG_ACTIVITY = 2
* OTHERS = 3
```

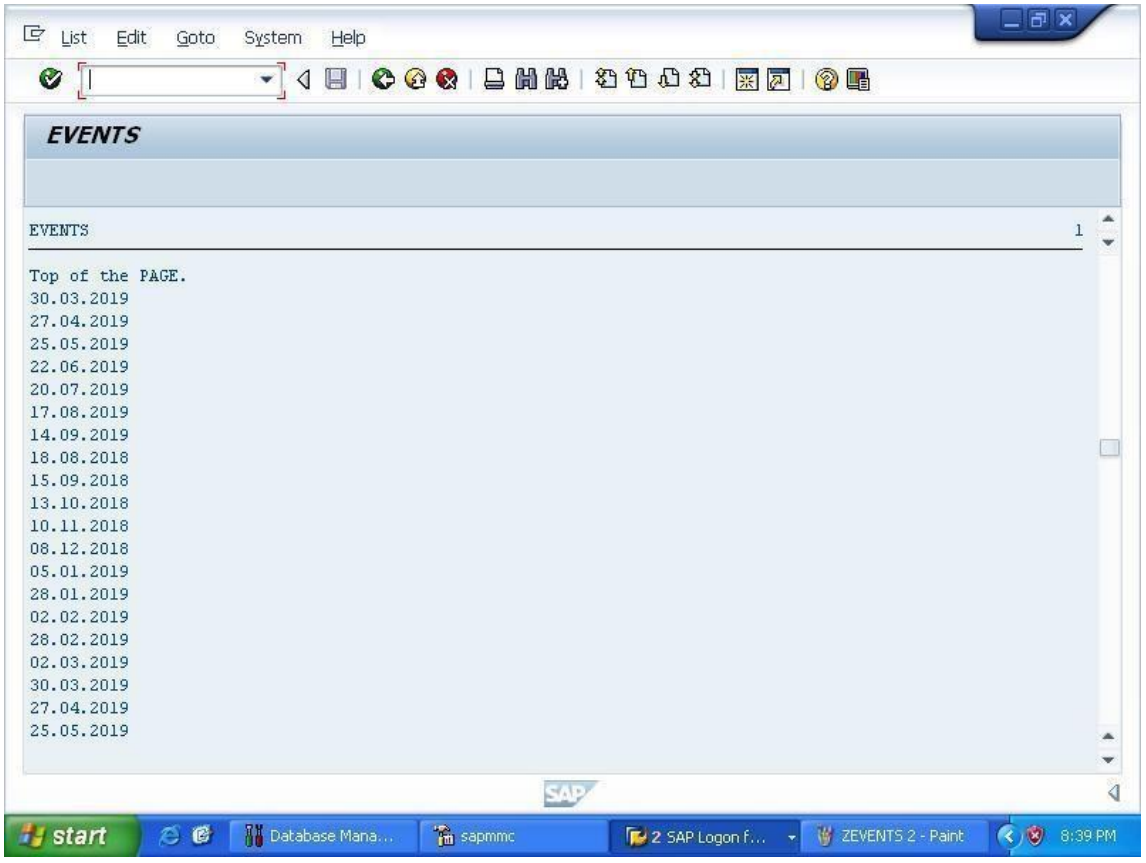
.

```
IF SY-SUBRC <> 0.

* Implement suitable error handling here

ENDIF.

START-OF-SELECTION.

SELECT * FROM SFLIGHT INTO gs_sfli.

WRITE: / gs_sfli-fldate.

ENDSELECT.

TOP-OF-PAGE.

WRITE: 'Top of the PAGE.'.
```

OUTPUT




List Edit Goto System Help
EVENTS
EVENTS 1
Top of the PAGE.
28.03.2019
25.04.2019
23.05.2019
20.06.2019
18.07.2019
15.08.2019
12.09.2019
15.08.2018
12.09.2018
10.10.2018
07.11.2018
05.12.2018
02.01.2019
30.01.2019
27.02.2019
27.03.2019
24.04.2019
22.05.2019
19.06.2019
17.07.2019
start
Database Mana...
sapmmc
2 SAP Logon f...
ZEVENTS 1 - Paint
8:39 PM


List Edit Goto System Help
EVENTS
EVENTS 1
Top of the PAGE.
30.03.2019
27.04.2019
25.05.2019
22.06.2019
20.07.2019
17.08.2019
14.09.2019
18.08.2018
15.09.2018
13.10.2018
10.11.2018
08.12.2018
05.01.2019
28.01.2019
02.02.2019
28.02.2019
02.03.2019
30.03.2019
27.04.2019
25.05.2019
start
Database Mana...
sapmmc
2 SAP Logon f...
ZEVENTS 2 - Paint
8:39 PM

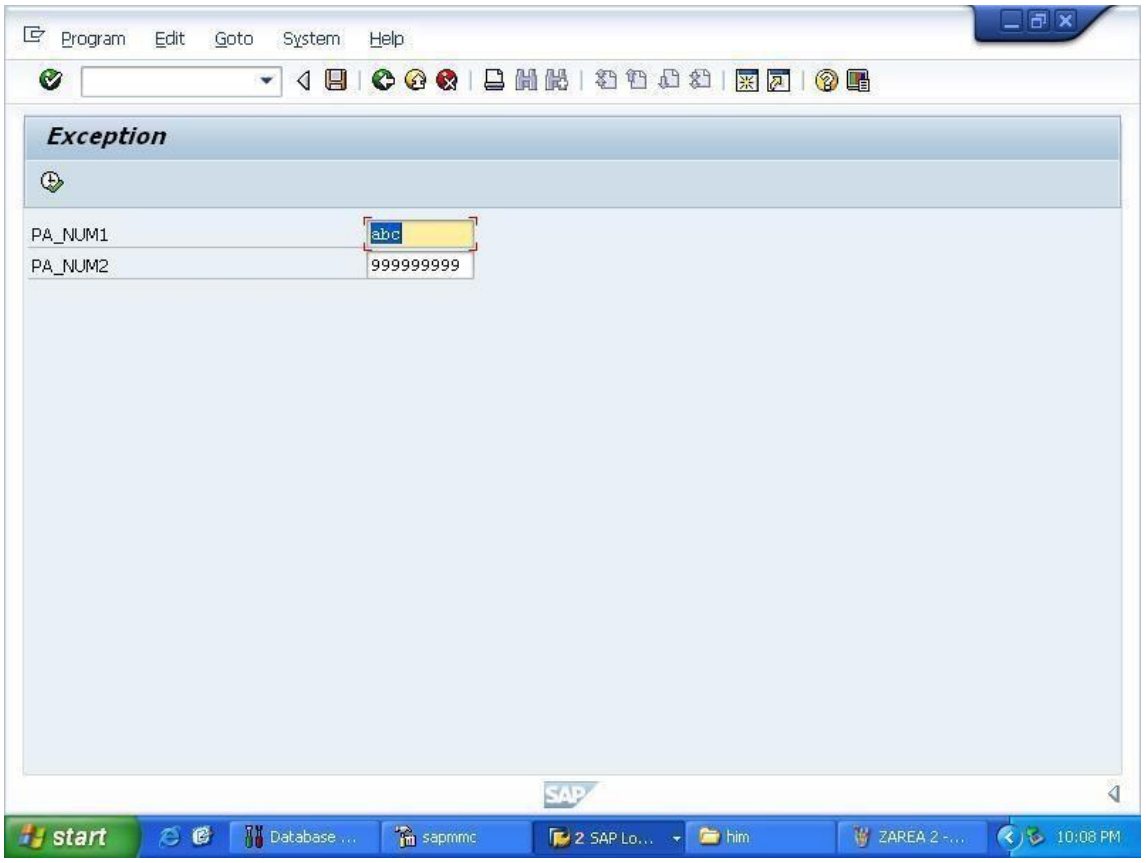
9.13 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS EXCEPTION HANDLING.

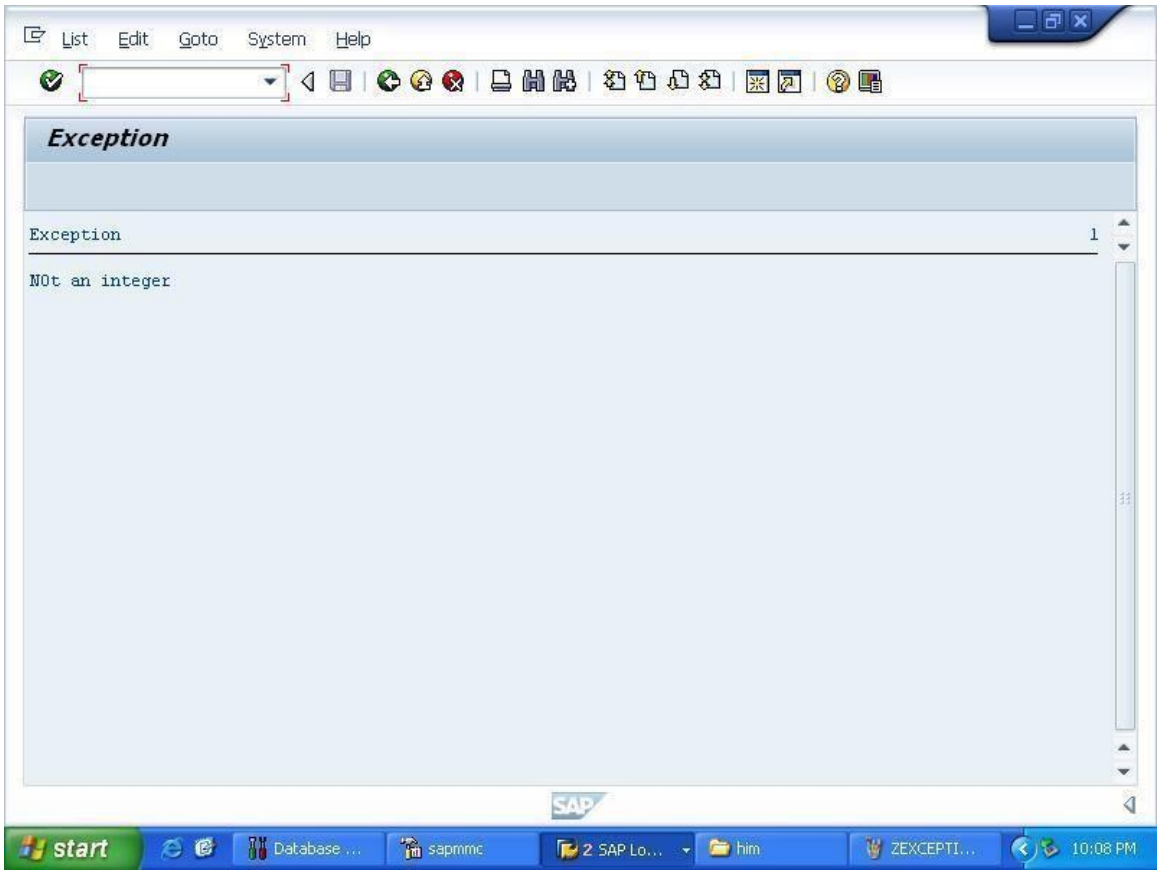
```
*&---------------------------------------------------------------------*
*& Report ZEXCEPTION
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZEXCEPTION.
PARAMETERS: pa_num1 TYPE c LENGTH 10 DEFAULT 'abc',
pa_num2 TYPE c LENGTH 10 DEFAULT '999999999'.
DATA: gv_num1 TYPE I,
gv_num2 TYPE I,
gv_result TYPE DECFLOAT34.
START-OF-SELECTION.
TRY.
gv_num1 = pa_num1.
gv_num2 = pa_num2.
gv_result = gv_num1 / gv_num2.
WRITE: /'Ressult is:', gv_result.
CATCH cx_sy_conversion_no_number.
WRITE: /'NOt an integer'.
CATCH cx_sy_conversion_overflow.
WRITE: /'Number too large'.
CATCH cx_sy_zerodivide.
```

WRITE: / 'Division is not possible.'.

ENDTRY.

OUTPUT





9.14 FLIGHT

&---

*& Report ZFLIGHT

*&

&---

*&

*&

&---

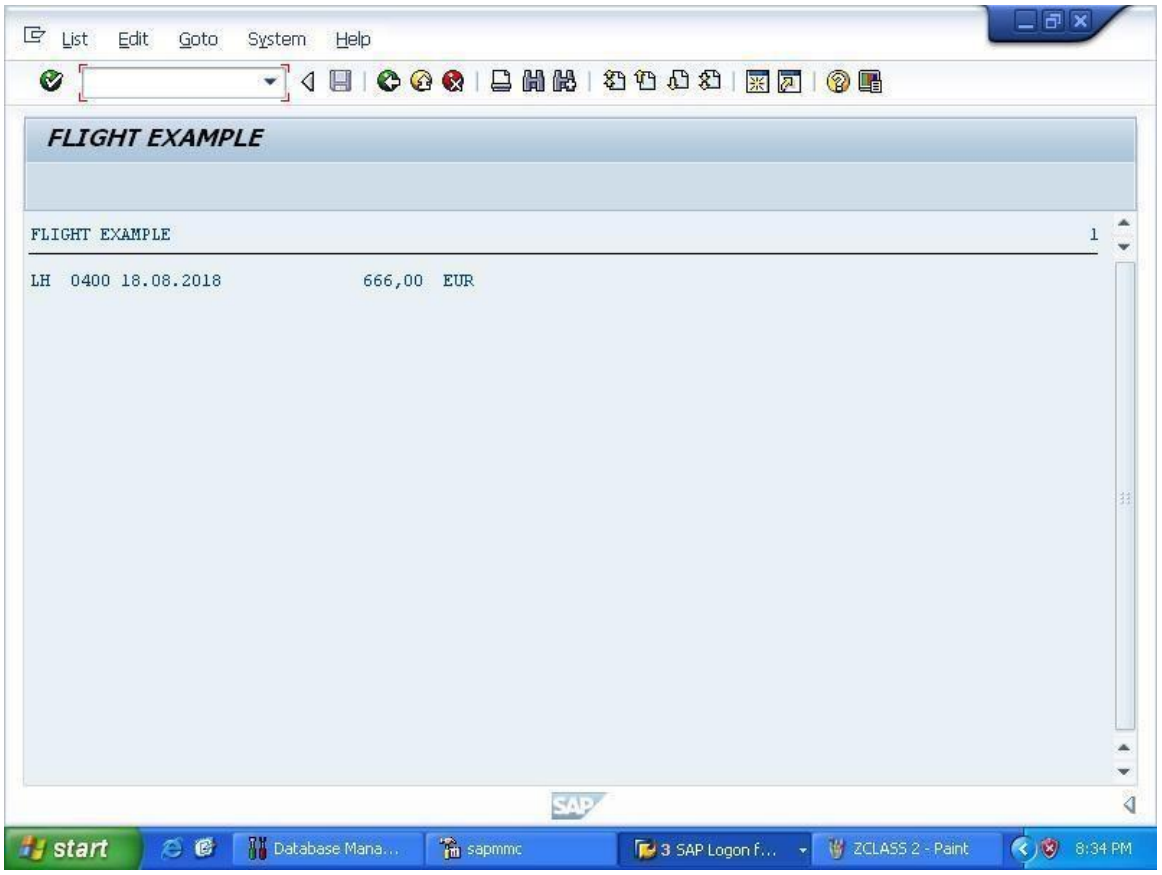
REPORT ZFLIGHT.

DATA FL_ELE TYPE SFLIGHT.

SELECT SINGLE * FROM SFLIGHT INTO FL_ELE WHERE CARRID = 'LH'.

WRITE: / FL_ELE-CARRID, FL_ELE-CONNID, FL_ELE-FLDATE, FL_ELE-PRICE, FL_ELE-CURRENCY.

OUTPUT

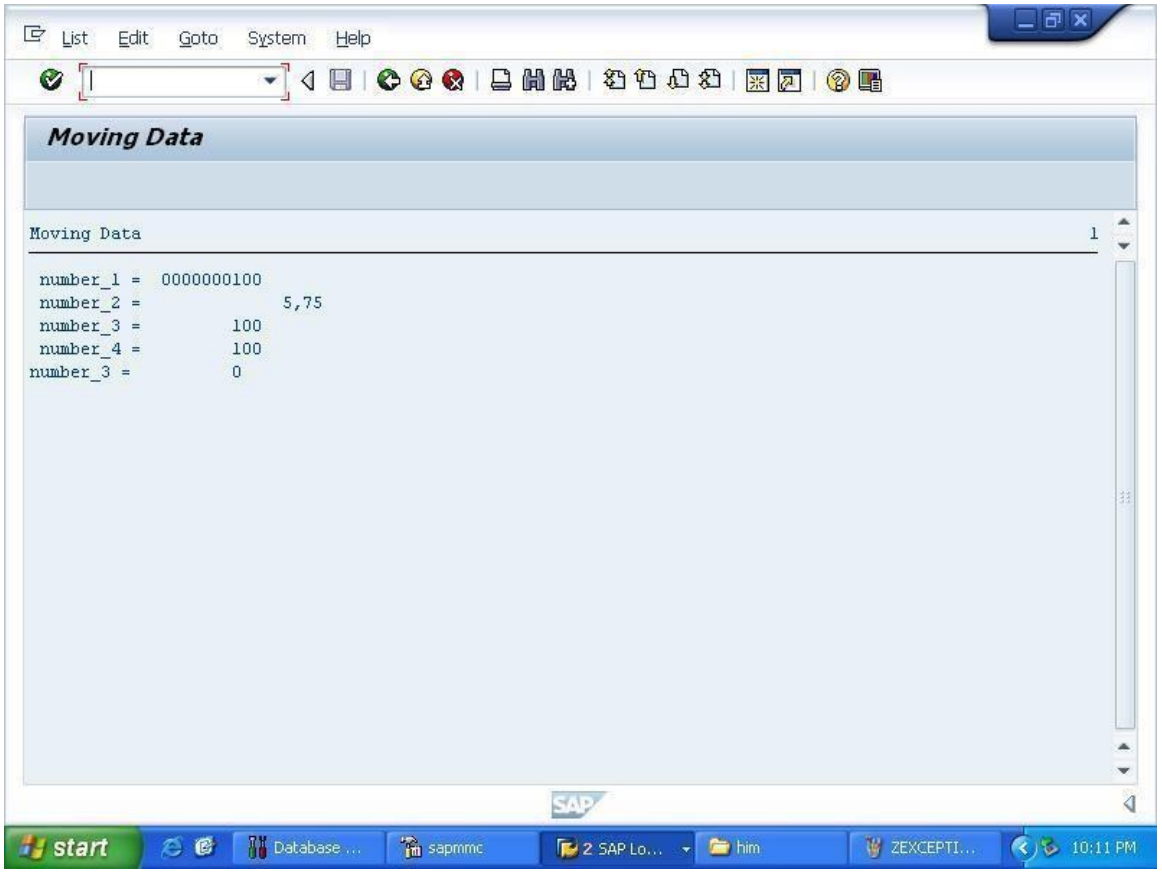


9.15 WRITE AN EXECUTABLE PROGRAM THAT MOVES DATA AMONG DIFFERENT DATA ELEMENTS.

```
*&---------------------------------------------------------------------*
*& Report ZHIM
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZHIM.
```

```
DATA : number_1(10) TYPE n,
number_2 TYPE P DECIMALS 2,
number_3 TYPE i,
number_4 LIKE number_3.
number_1 = 100 .
MOVE '5.75' TO number_2.
number_3 = number_1.
WRITE : / ' number_1 = ',number_1.
WRITE : / ' number_2 = ',number_2.
Write : / ' number_3 = ', number_3.
MOVE number_3 TO number_4.
write : / ' number_4 = ', number_4.
CLEAR number_3.
Write : / 'number_3 =',number_3.
```

OUTPUT



9.16 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS THE CONCEPT OF INTERFACES.

&---

*& Report ZINTERFACE

*&

&---

*&

*&

&---

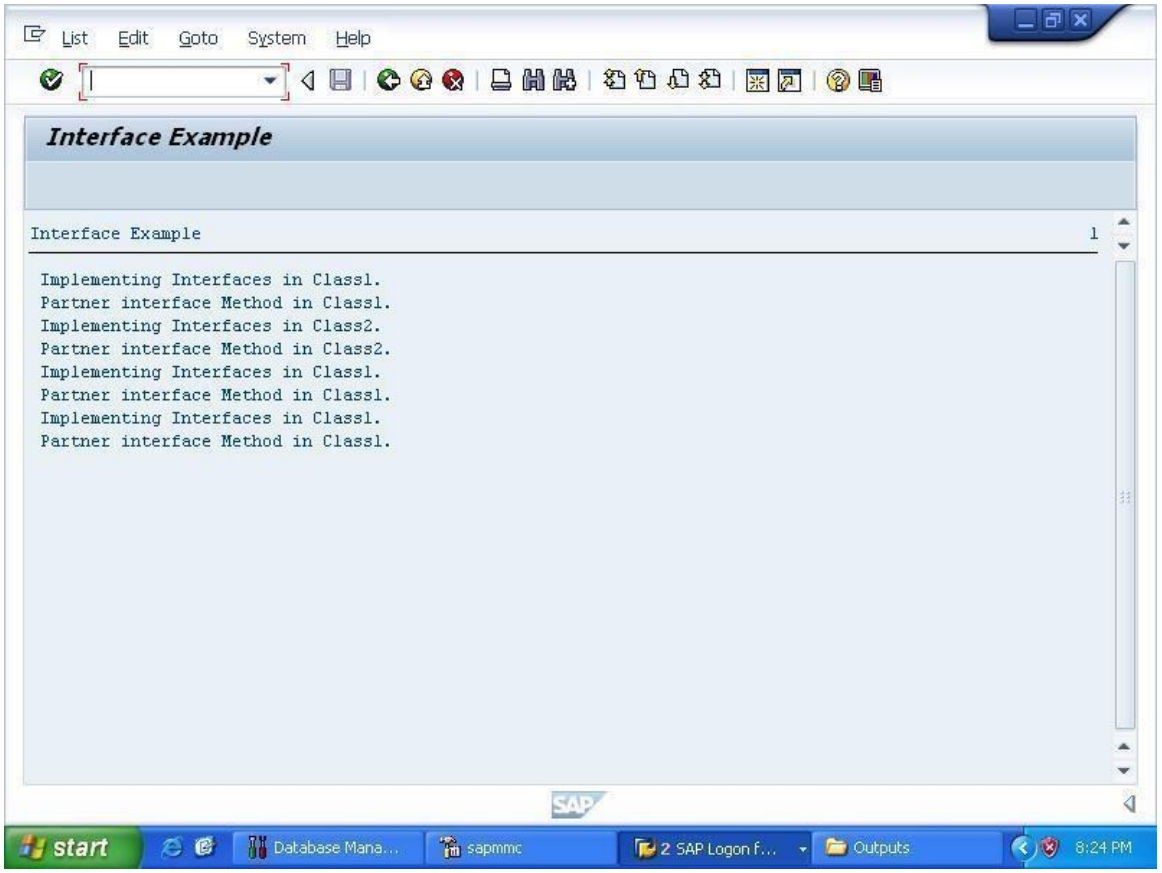
REPORT ZINTERFACE.

INTERFACE partner.

```
METHODS display_partner.
ENDINTERFACE.
INTERFACE service.
INTERFACES partner.
DATA text(35).
METHODS display.
ENDINTERFACE.
CLASS lcl_rental1 DEFINITION.
PUBLIC SECTION.
INTERFACES service.
ENDCLASS.
CLASS lcl_rental1 IMPLEMENTATION.
METHOD partner~display_partner.
WRITE: / ' Partner interface Method in Class1. '.
ENDMETHOD.
METHOD service~display.
service~text = ' Implementing Interfaces in Class1. '.
WRITE: / service~text.
ENDMETHOD.
ENDCLASS.
CLASS lcl_rental2 DEFINITION.
PUBLIC SECTION.
INTERFACES service.
ENDCLASS.
CLASS lcl_rental2 IMPLEMENTATION.
METHOD partner~display_partner.
```

```
WRITE: / ' Partner interface Method in Class2. '.
ENDMETHOD.
METHOD service~display.
WRITE: / ' Implementing Interfaces in Class2. '.
ENDMETHOD.
ENDCLASS.
START-OF-SELECTION.
DATA: ref1 TYPE REF TO lcl_rental1,
ref2 TYPE REF TO lcl_rental1,
ref3 TYPE REF TO lcl_rental2,
go_service TYPE REF TO service,
go_partner TYPE REF TO partner.
CREATE OBJECT ref1.
CREATE OBJECT ref3.
ref1->service~display( ).
ref1->partner~display_partner( ).
ref3->service~display( ).
ref3->partner~display_partner( ).
go_service = ref1.
go_service->display( ).
go_partner = ref1.
go_partner->display_partner( ).
ref2 ?= go_service.
ref2->service~display( ).
ref2->partner~display_partner( ).
```

OUTPUT



9.17 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS VARIOUS OPERATIONS ON INTERNAL TABLES.

&---

*& Report ZINTERNALTABLE

*&

&---

*&

*&

&---

REPORT ZINTERNALTABLE.

TYPES: begin of student,

```
introllno TYPE c LENGTH 10 ,
name type c LENGTH 10 ,
marks type c LENGTH 10,
end of student.
DATA: teacher TYPE STANDARD TABLE OF student
WITH NON-UNIQUE KEY introllno.
data:student1 type student.
student1-introllno = '100'.
student1-name = 'anand'.
student1-marks = '45'.
data:student2 TYPE student.
student2-introllno = '150'.
student2-name = 'himanshu'.
student2-marks = '99'.
DATA: s3 type student.
WRITE:/'before insertion'.
insert Student1 into TABLE teacher.
insert Student2 into TABLE teacher.
insert Student1 into TABLE teacher.
loop at teacher into s3.
write: /'the table is',s3-name,
s3-introllno,s3-marks.
ENDLOOP.
write:/'before appending'.
APPEND student2 to teacher.
loop at teacher into s3.
```

```
write:/ 'the table is',s3-name,
s3-introllno,s3-marks.
ENDLOOP.
data: student3 TYPE student.
student3-introllno = '120'.
student3-name = 'basnayat'.
student3-marks = '990'.
Append student3 to teacher.
READ TABLE teacher into s3 WITH TABLE KEY introllno = '120'.
write:/ 'using read starement the record is',s3-name,
s3-introllno,s3-marks.
loop at teacher into s3.
write:/ 'before deleting the table is',s3-name,
s3-introllno,s3-marks.
ENDLOOP.
delete teacher where introllno = '120'.
loop at teacher into s3.
write:/ 'the final table is',s3-name,
s3-introllno,s3-marks.
ENDLOOP.
*& REFRESH teacher.
* if sy-subrc=0.
* write:/'table contain some data'.
* else.
* write:/'table has been refreshed'.
* ENDIF.
```

* insert Student1 into TABLE teacher.

* insert Student2 into TABLE teacher.

*free teacher.

* if sy-subrc = 0.

* write:/'table contain some data'.

* else.

* write:/'table has been refreshed'.

* ENDIF.

* insert Student1 into TABLE teacher.

* insert Student2 into TABLE teacher.

* clear teacher.

* if sy-subrc = 0.

* write:/'table contain some data'.

* else.

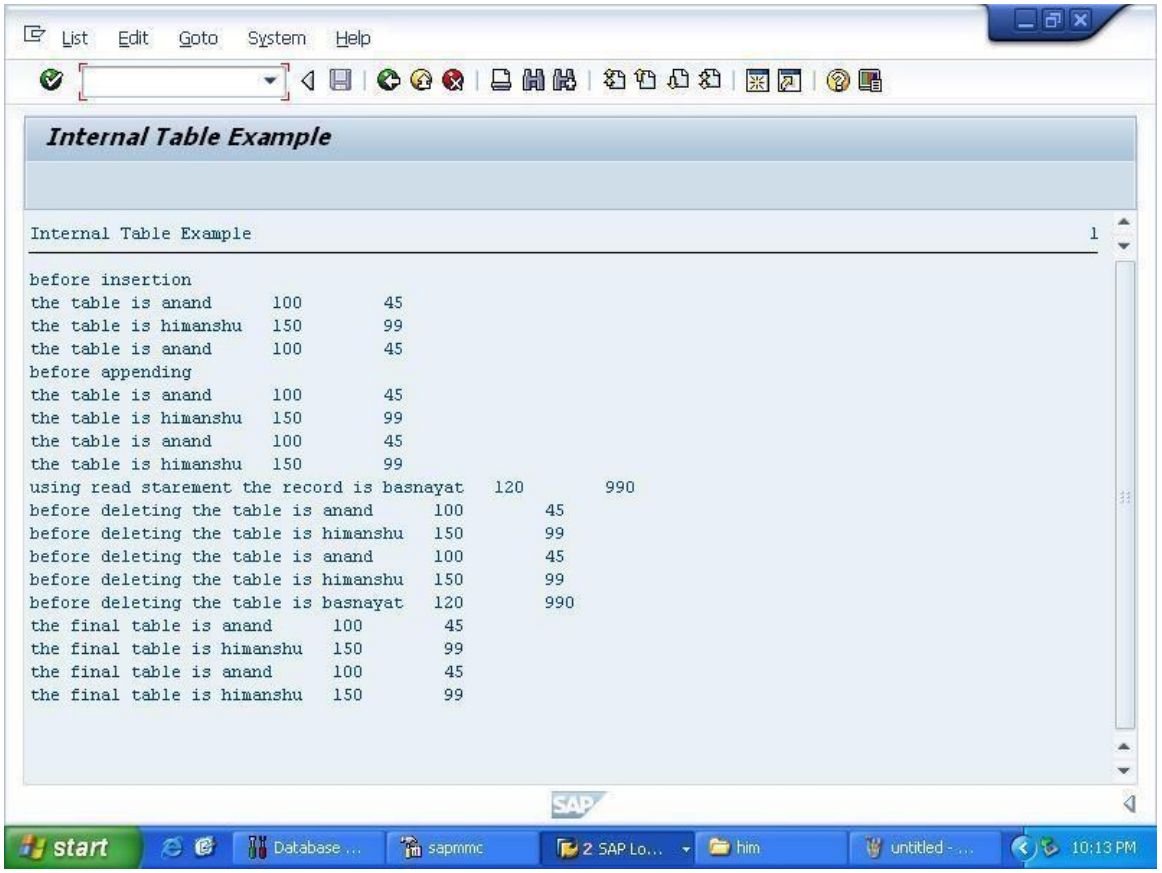
* write:/'table has been refreshed'.

* ENDIF.

* insert Student1 into TABLE teacher. insert Student2 into TABLE teacher.

* MODIFY TABLE teacher from student3 TRANSPORTING introllno = '100' .

OUTPUT

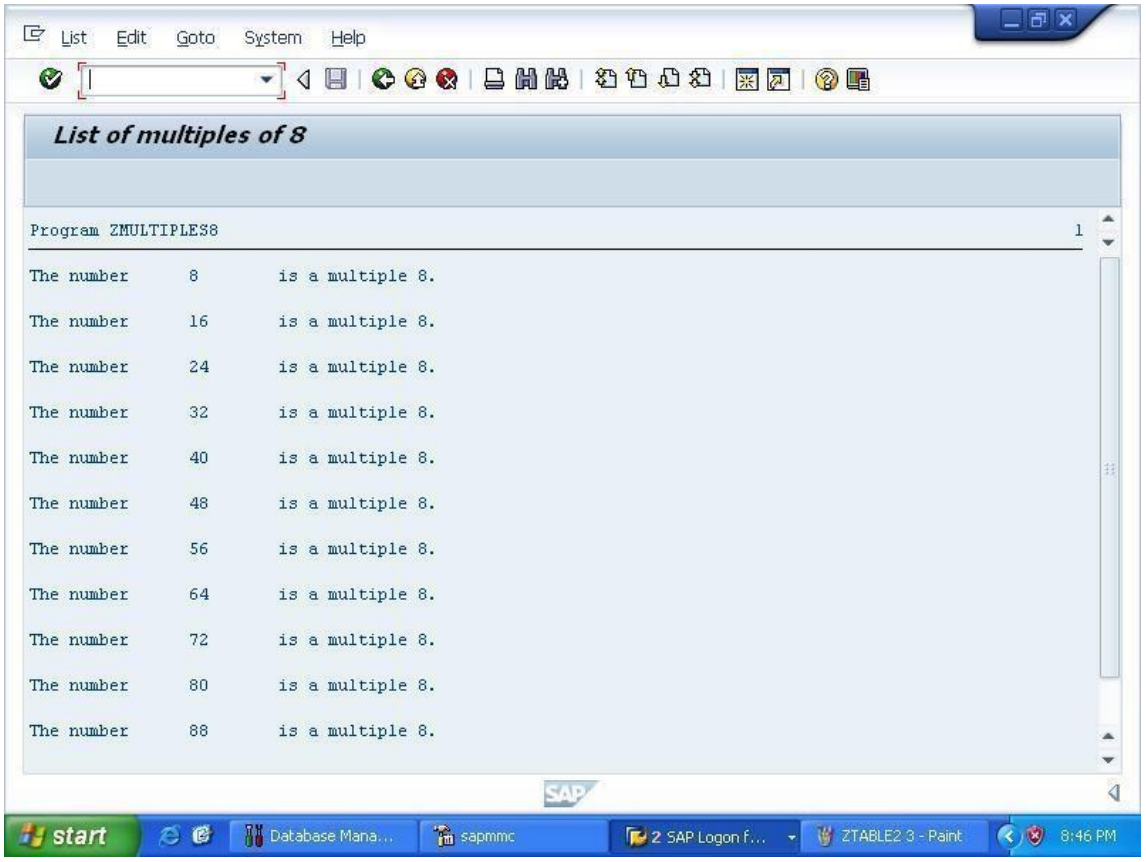


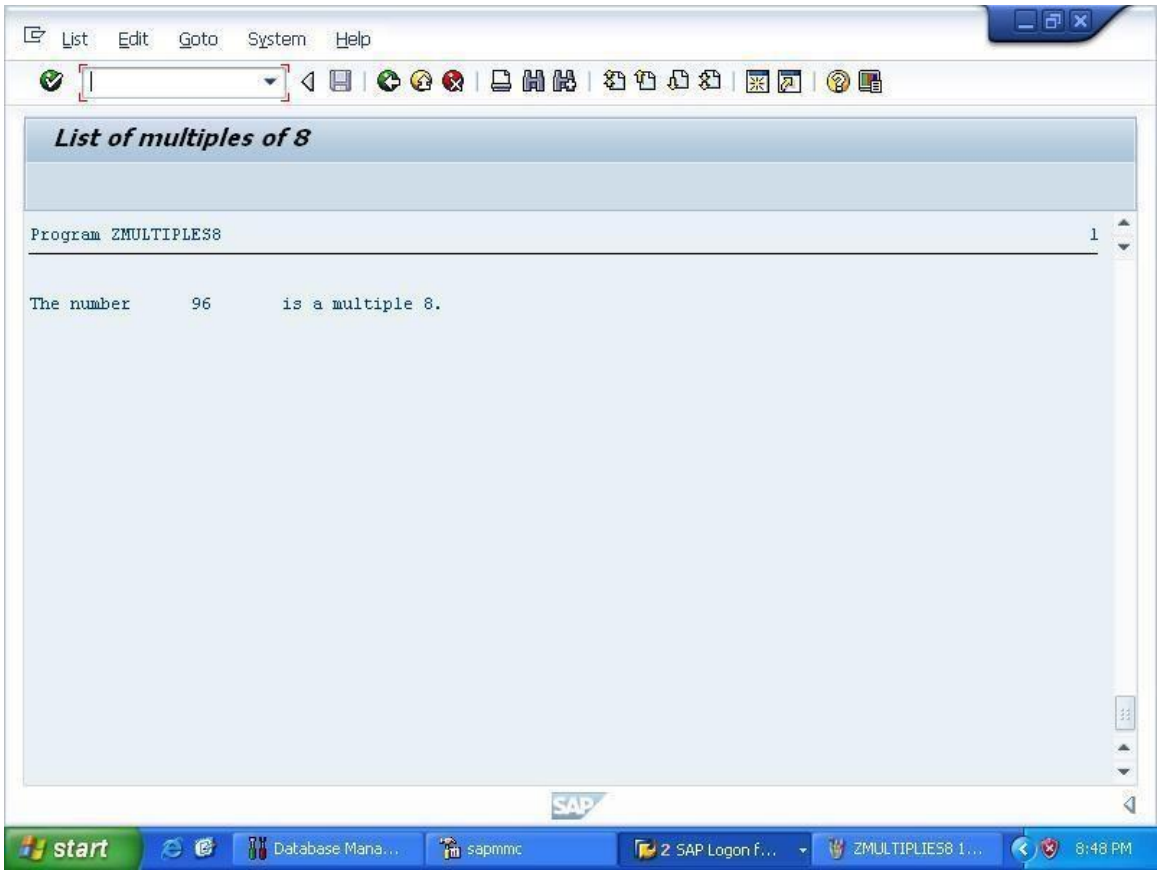
9.18 WRITE AN EXECUTABLE PROGRAM THAT COUNTS FROM 1 TO 100 AND FOR EACH MULTIPLE OF 8, WRITE THE MESSAGE: "THE NUMBER [NUMBER] IS A MULTIPLE OF 8."

```
*&---------------------------------------------------------------------*
*& Report ZMULTIPLES8
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZMULTIPLES8.
```

```
SY-TITLE = ' List of multiples of 8 '.

DATA Y TYPE I.

DO 100 TIMES.

Y = SY-INDEX MOD 8.

IF ( Y = 0 ).

WRITE: 'The number ', SY-INDEX CENTERED, ' is a multiple 8.'.

SKIP.

ENDIF.

ENDDO..
```

OUTPUT





9.19 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS VARIOUS TYPES OF STRUCTURES AND THEIR NESTING.

&---

*& Report ZNESTED_STRUCTURE_EX

*&

&---

*&

*&

&---

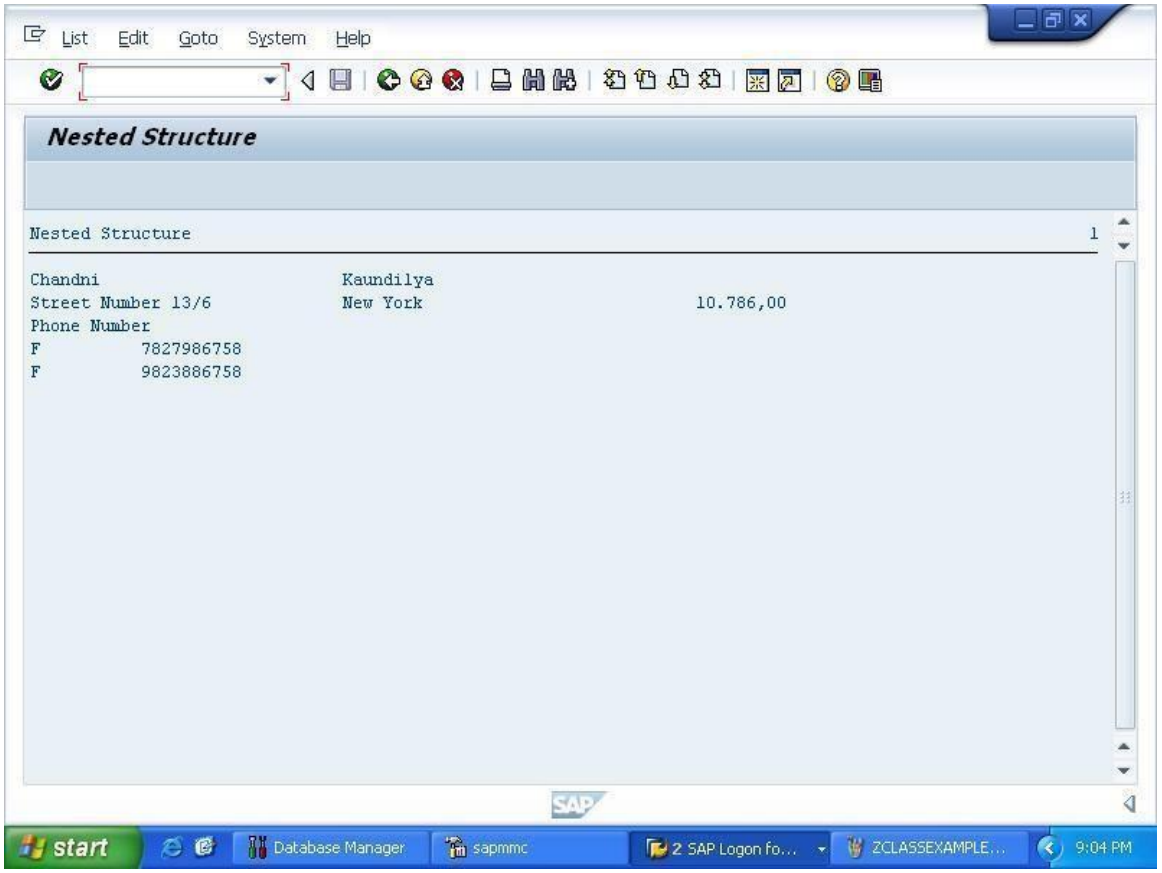
REPORT ZNESTED_STRUCTURE_EX.

DATA: wa_pers TYPE ZPERSON1,

wa_phone TYPE ZPHONE.

```
START-OF-SELECTION.
wa_pers-name-firstname = 'Chandni'.
wa_pers-name-lastname = 'Kaundilya'.
wa_pers-street = 'Street Number 13/6'.
wa_pers-zipcode = 10786.
wa_pers-city = 'New York'.
wa_phone-phonetype = 'F'.
wa_phone-phonenumber = 7827986758.
INSERT wa_phone INTO TABLE wa_pers-phoneno.
wa_phone-phonetype = 'F'.
wa_phone-phonenumber = 9823886758.
INSERT wa_phone INTO TABLE wa_pers-phoneno.
WRITE: / wa_pers-name-firstname, wa_pers-name-lastname.
WRITE: / wa_pers-street, wa_pers-city, wa_pers-zipcode.
WRITE: / 'Phone Number'.
LOOP AT wa_pers-phoneno INTO wa_phone.
WRITE: / wa_phone-phonetype, wa_phone-phonenumber.
ENDLOOP.
```

OUTPUT



9.20 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS VARIOUS TYPES OF PARAMETERS.

&---

*& Report ZPARAMETER

*&

&---

*&

*&

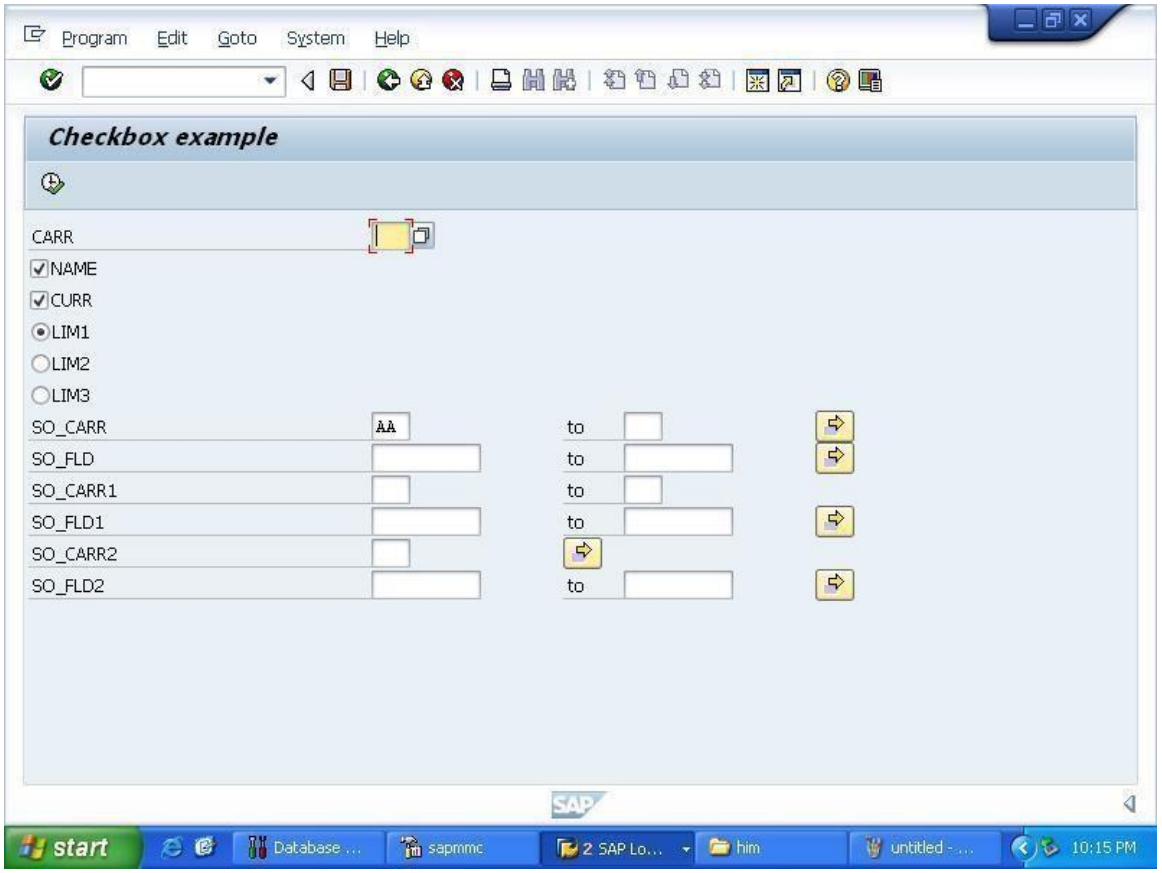
&---

REPORT ZPARAMETER.

PARAMETERS: carr TYPE sflight-carrid,

```
name As Checkbox DEFAULT 'x',
curr as checkbox default 'x',
lim1 RADIOBUTTON GROUP lim DEFAULT 'X',
lim2 RADIOBUTTON GROUP lim,
lim3 RADIOBUTTON GROUP lim.
CONSTANTS gc_mark VALUE 'x'.
DATA: gs_flight TYPE sflight.
SELECT-OPTIONS : so_carr FOR gs_flight-carrid DEFAULT 'AA',
so_fld FOR gs_flight-fldate.
SELECT-OPTIONS : so_carr1 FOR gs_flight-carrid NO-EXTENSION,
so_fld1 FOR gs_flight-fldate.
SELECT-OPTIONS : so_carr2 FOR gs_flight-carrid NO INTERVALS,
so_fld2 FOR gs_flight-fldate.
```

OUTPUT



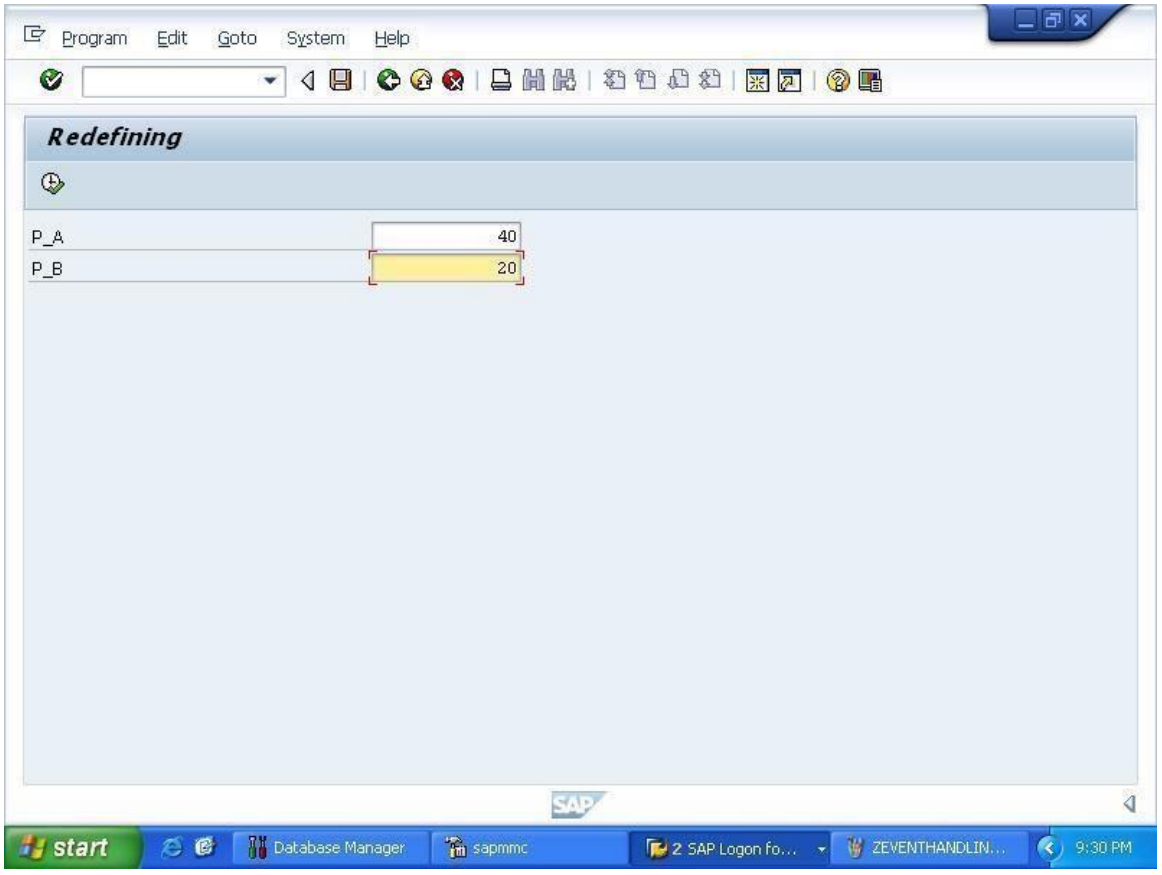
9.21 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS THE CONCEPT OF INHERITANCE ALONG WITH REDEFINITION OF SUPER CLASS METHODS.

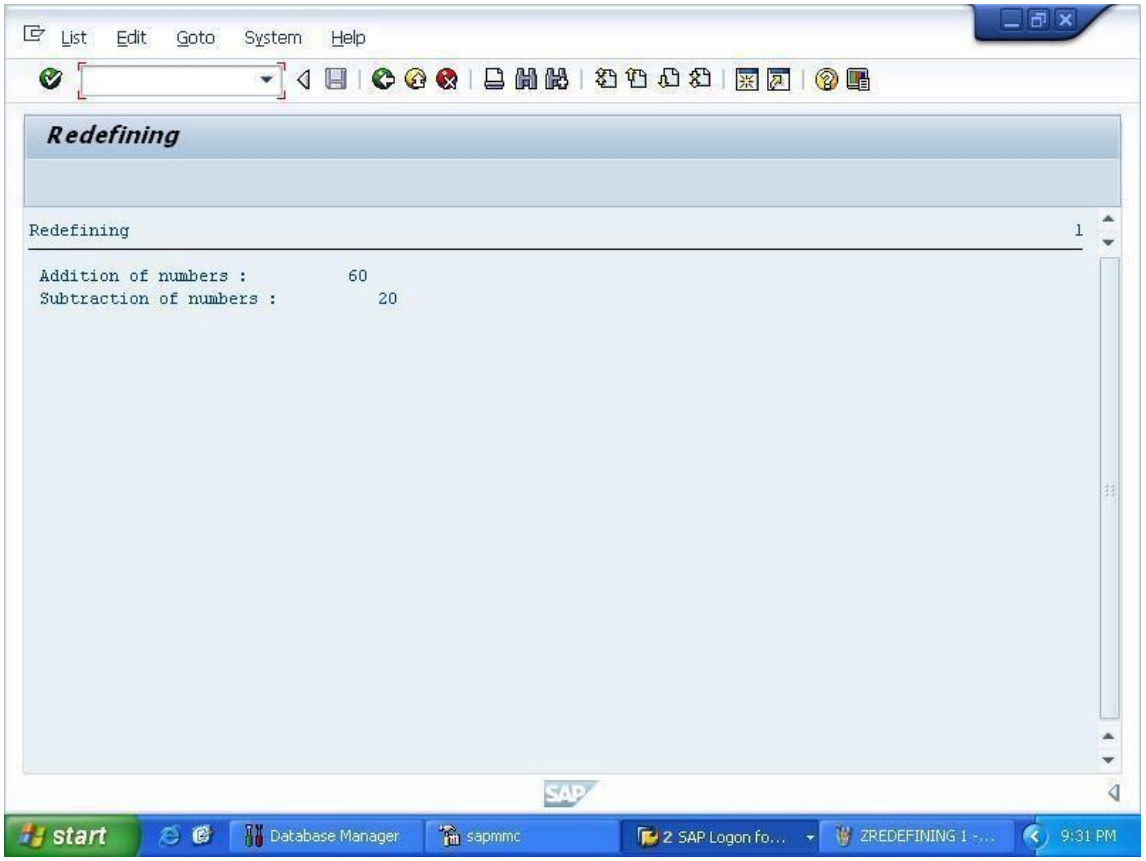
```
*&---------------------------------------------------------------------*
*& Report ZREDEFINING
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZREDEFINING.
```

```
CLASS super_class DEFINITION.
PUBLIC SECTION.
METHODS: Addition1 IMPORTING g_a TYPE I
g_b TYPE I
EXPORTING g_c TYPE I.
ENDCLASS.
CLASS super_class IMPLEMENTATION.
METHOD Addition1.
g_c = g_a +  g_b.
ENDMETHOD.
ENDCLASS.
CLASS sub_class DEFINITION INHERITING FROM super_class.
PUBLIC SECTION.
METHODS: Addition1 REDEFINITION.
ENDCLASS.
CLASS sub_class IMPLEMENTATION.
METHOD Addition1.
CALL METHOD super->Addition1
EXPORTING
g_a = g_a
g_b = g_b
IMPORTING
g_c = g_c.
WRITE: ' Addition of numbers : ' , g_c.
g_c = g_a -  g_b.
ENDMETHOD.
```

```
ENDCLASS.

START-OF-SELECTION.

PARAMETERS: p_a TYPE I,

p_b TYPE I.

DATA: addition TYPE I,

ref1 TYPE REF TO sub_class.

CREATE OBJECT ref1.

CALL METHOD ref1->Addition1

EXPORTING

g_a = p_a

g_b = p_b

IMPORTING

g_c = addition.

WRITE: / ' Subtraction of numbers : ' , addition.
```

OUTPUT


Program Edit Goto System Help
Redefining
P_A 40
P_B 20
start Database Manager sapmmc 2 SAP Logon fo... ZEVENTHANDLIN... 9:30 PM


List Edit Goto System Help
Redefining
Redefining 1
Addition of numbers : 60
Subtraction of numbers : 20
start Database Manager sapmmc 2 SAP Logon fo... ZREDEFINING 1 -... 9:31 PM

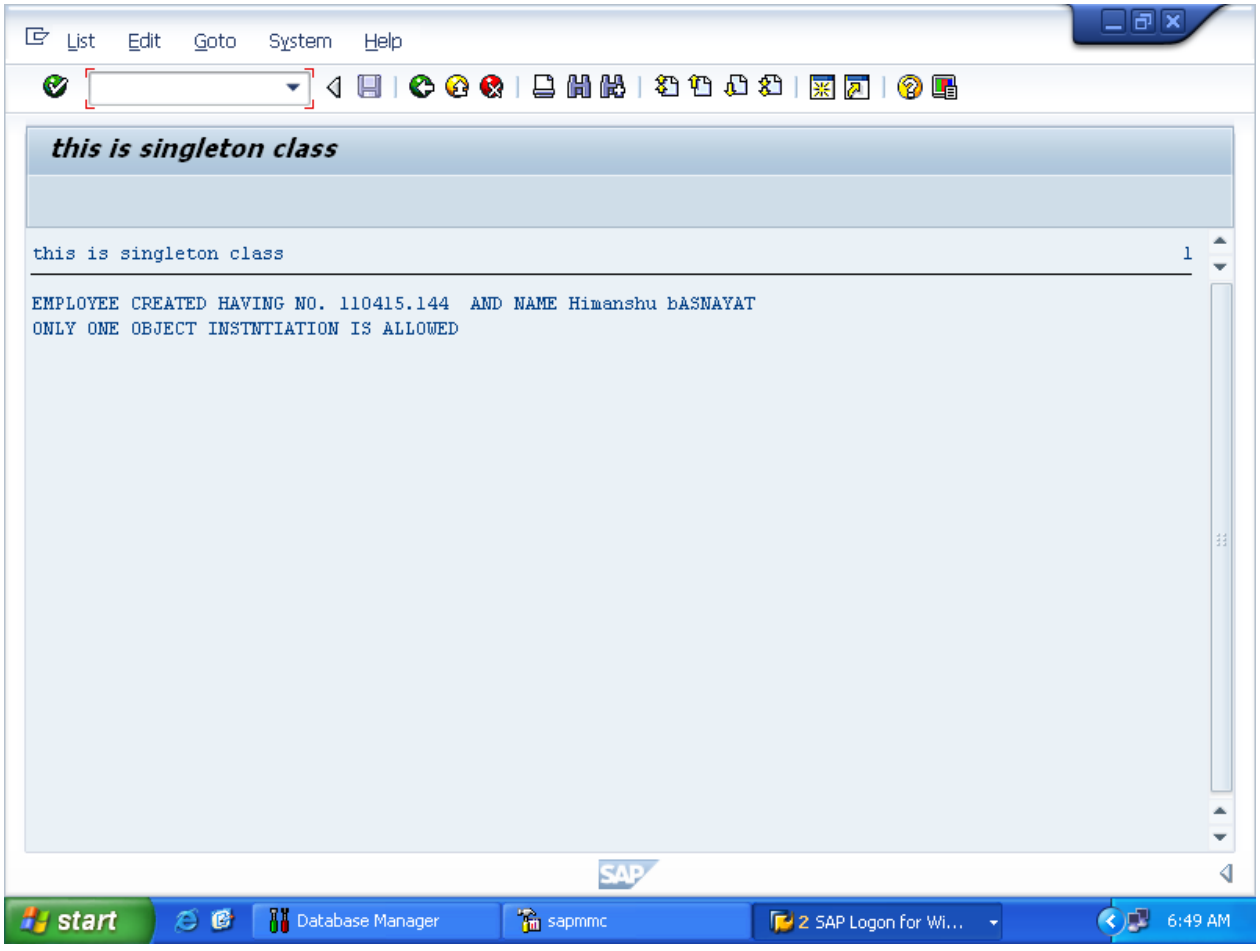
9.22 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS SINGLETON PATTERN.

```
*&---------------------------------------------------------------------*
*& Report ZSINGLETON
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZSINGLETON.
CLASS singleton_class DEFINITION CREATE PRIVATE.
PUBLIC SECTION.
CLASS-METHODS
factory IMPORTING number TYPE i
employee_name TYPE String
EXPORTING employee_obj TYPE REF TO singleton_class.
METHODS constructor IMPORTING number TYPE I
employee_name TYPE String.
PRIVATE SECTION.
DATA: employee_number TYPE i,
employee_name TYPE String.
CLASS-DATA: number_of_instances TYPE I.
ENDCLASS.
CLASS singleton_class IMPLEMENTATION.
METHOD factory.
IF number_of_instances EQ 0.
```

```
CREATE OBJECT employee_obj
EXPORTING
number = number
employee_name = employee_name.
number_of_instances = 1.
ELSE.
WRITE: / 'Only one object instantiated is allowed.'.
ENDIF.
ENDMETHOD.
METHOD constructor.
me->employee_number = number.
me->employee_name = employee_name.
WRITE: / 'Employee Created having no.', number, 'and name ',employee_name.
ENDMETHOD.
ENDCLASS.
START-OF-SELECTION.
DATA: obj TYPE REF TO singleton_class.
DATA: number TYPE I,
name TYPE String.
number = '110457834'.
name = 'Chandni'.
CALL METHOD singleton_class=>factory
EXPORTING
number = number
employee_name = name
IMPORTING
```

```
employee_obj = obj.

DATA: obj2 TYPE REF TO singleton_class.

number = '57492801'.

name = 'Ritu'.

CALL METHOD singleton_class=>factory

EXPORTING

number = number

employee_name = name

IMPORTING

employee_obj = obj2.
```

OUTPUT

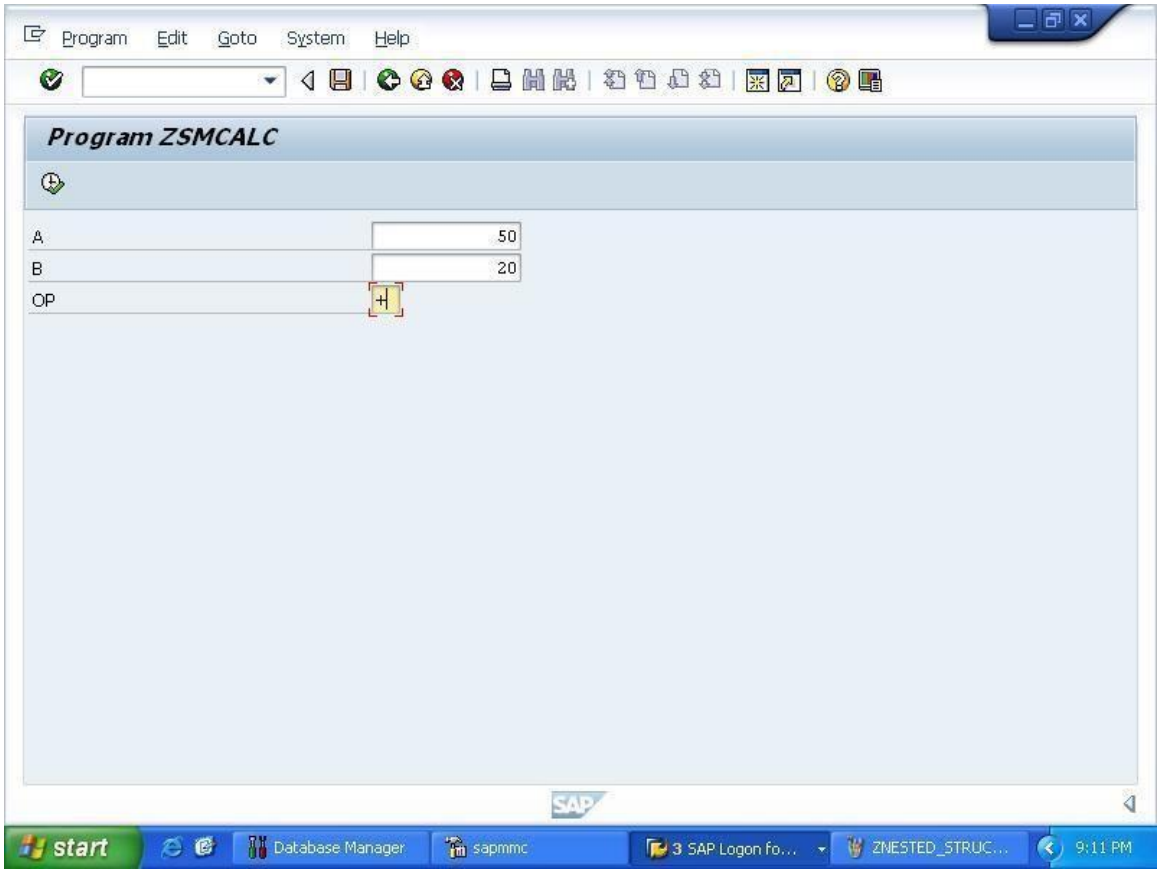


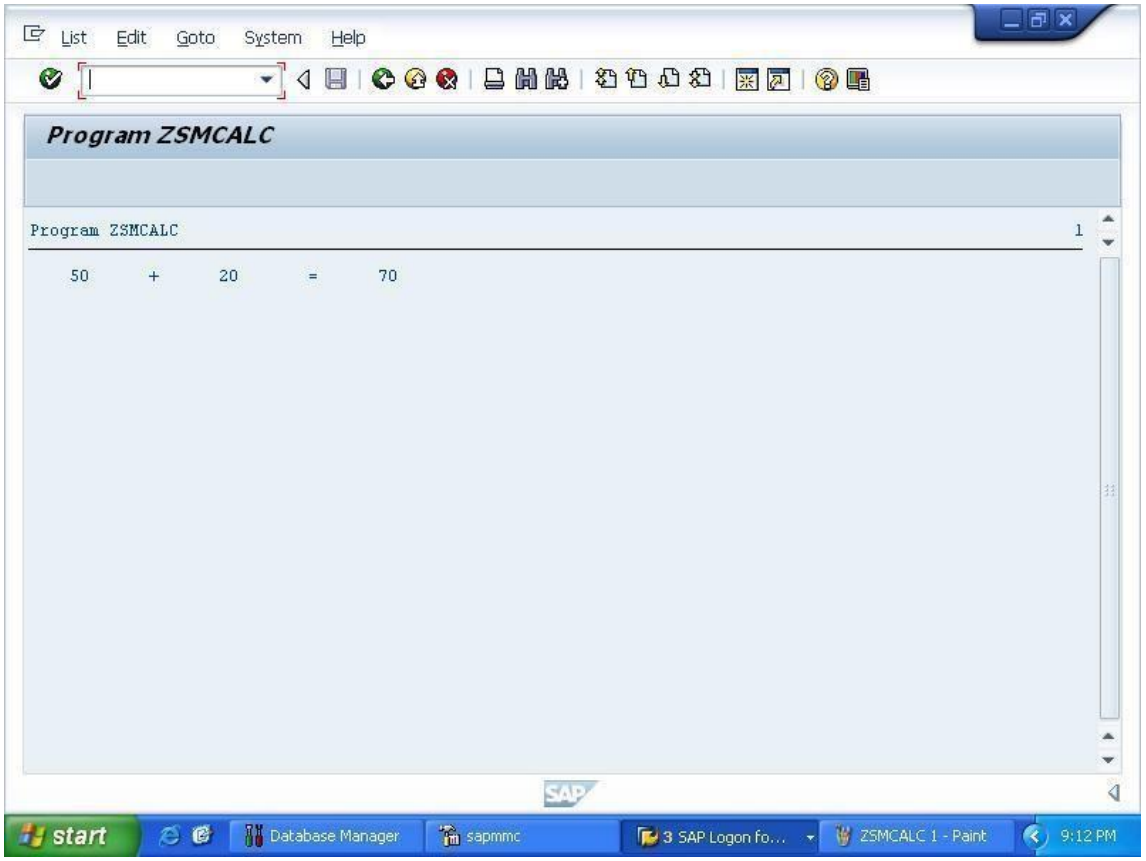
9.23 WRITE AN EXECUTABLE PROGRAM THAT GETS TWO INTEGERS INSIDE VARIABLES AND PERFORM THE ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION AND POWER BETWEEN THEM.

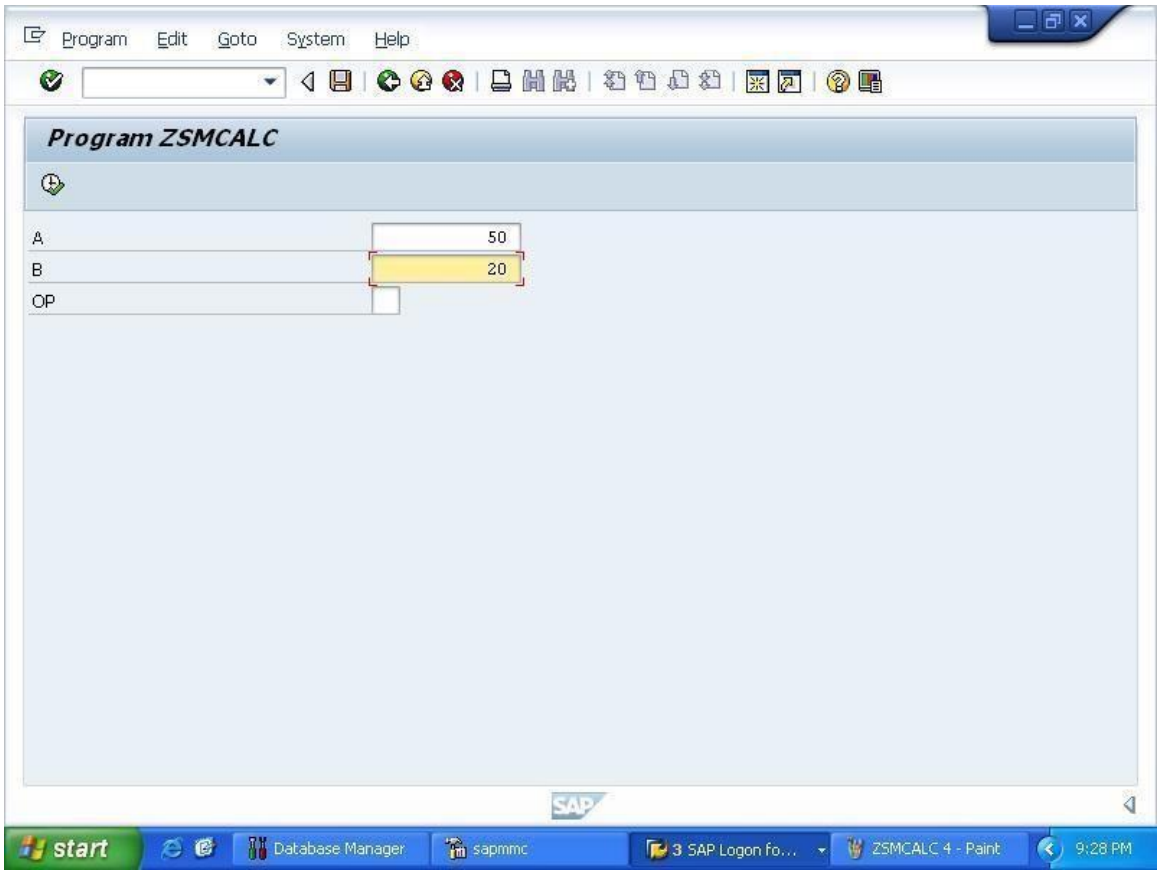
```
*&---------------------------------------------------------------------*
*& Report ZSMCALC
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZSMCALC.
PARAMETERS: A TYPE I,
B TYPE I,
OP TYPE C LENGTH 2.
DATA: R TYPE I VALUE 0,
RESULT TYPE P.
IF ( OP = '+' OR OP = '-' OR OP = '*' OR OP = '/' OR
OP = '**' AND B <> 0 ).
CASE  OP.
WHEN  '+'.
R = A + B.
WRITE: A CENTERED, OP CENTERED,
B CENTERED, ' = ', R CENTERED.
WHEN '-'.
R = A - B.
```

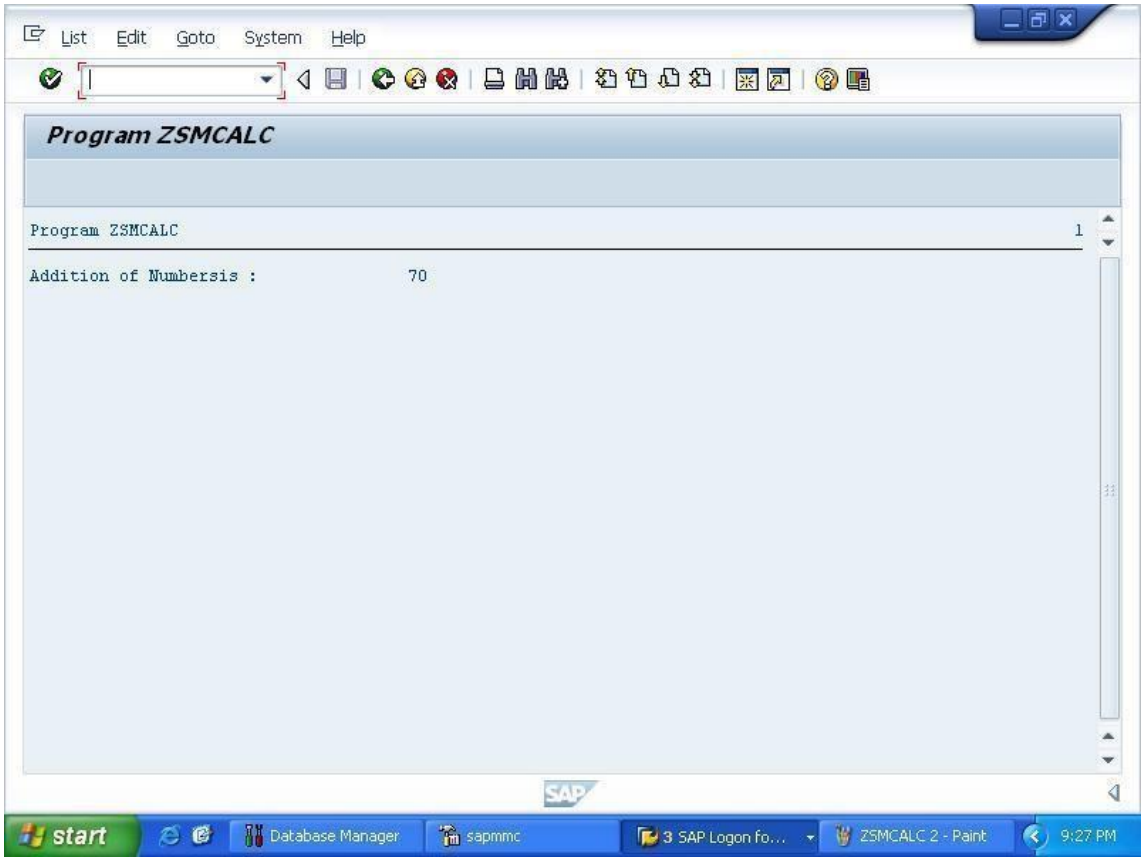
```
WRITE: A CENTERED, OP CENTERED,
B CENTERED, ' = ', R CENTERED.
WHEN '*'.
R = A * B.
WRITE: A CENTERED, OP CENTERED,
B CENTERED, ' = ', R CENTERED.
WHEN '/'.
R = A / B.
WRITE: A CENTERED, OP CENTERED,
B CENTERED, ' = ', R CENTERED.
WHEN '**'.
R = A ** B.
WRITE: A CENTERED, OP CENTERED,
B CENTERED, ' = ', R CENTERED.
WHEN OTHERS.
WRITE 'Operator undefined...!!'.
ENDCASE.
ELSE.
CALL FUNCTION 'ZSUM'
EXPORTING
PA_INT1 = A
PA_INT2 = B
IMPORTING
GV_RESULT = RESULT.
ENDIF.
```

OUTPUT


Program Edit Goto System Help
Program ZSMCALC
A 50
B 20
OP +
start Database Manager sapmmc 3 SAP Logon fo... ZNESTED_STRUC... 9:11 PM


List Edit Goto System Help
Program ZSMCALC
Program ZSMCALC 1
50 + 20 = 70
start Database Manager sapmmc 3 SAP Logon fo... ZSMCALC 1 - Paint 9:12 PM


Program Edit Goto System Help
Program ZSMCALC
A 50
B 20
OP
start Database Manager sapmmc 3 SAP Logon fo... ZSMCALC 4 - Paint 9:28 PM


List Edit Goto System Help
Program ZSMCALC
Program ZSMCALC 1
Addition of Numbersis : 70
start Database Manager sapmmc 3 SAP Logon fo... ZSMCALC 2 - Paint 9:27 PM

9.24 WRITE AN EXECUTABLE PROGRAM THAT IMPLEMENTS STRUCTURES AND INTERNAL TABLES.

```
*&---------------------------------------------------------------------*
*& Report ZSTRAB
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZSTRAB.
TYPES: BEGIN OF STUD,
ROLLN TYPE I,
NAME TYPE STRING,
MARKS TYPE P,
END OF STUD.
DATA: STUD1 TYPE STUD,
STUDENT TYPE SORTED TABLE OF STUD WITH UNIQUE KEY ROLLN,
STUD3 TYPE STUD.
**STUD3 LIKE LINE OF STUD
**STUD2 TYPE ZSELF,
**TEACH TYPE ZSELC,
STUD1-ROLLN = 1001.
STUD1-NAME = 'HEENA'.
STUD1-MARKS = '70.50'.
INSERT STUD1 INTO TABLE STUDENT.
STUD1-ROLLN = 1002.
```

STUD1-NAME = 'PRIYA'.

STUD1-MARKS = '74.50'.

INSERT STUD1 INTO TABLE STUDENT.

STUD1-ROLLN = 1003.

STUD1-NAME = 'KRITIKA'.

STUD1-MARKS = '80.50'.

INSERT STUD1 INTO TABLE STUDENT.

STUD1-ROLLN = 1004.

STUD1-NAME = 'ASTHA'.

STUD1-MARKS = '76.50'.

INSERT STUD1 INTO TABLE STUDENT.

LOOP AT STUDENT INTO STUD3.

WRITE : / STUD3-ROLLN LEFT-JUSTIFIED, / STUD3-NAME LEFT-JUSTIFIED, / STUD3-MARKS LEFT-JUSTIFIED.

SKIP.

ENDLOOP.

*WRITE : / STUD1-ROLLN LEFT-JUSTIFIED, / STUD1-NAME RIGHT-JUSTIFIED, / STUD1-MARKS LEFT-JUSTIFIED.

**SKIP.

**STUD2-ROLL_NO = 1002.

**STUD2-NAME = 'PRIYA'.

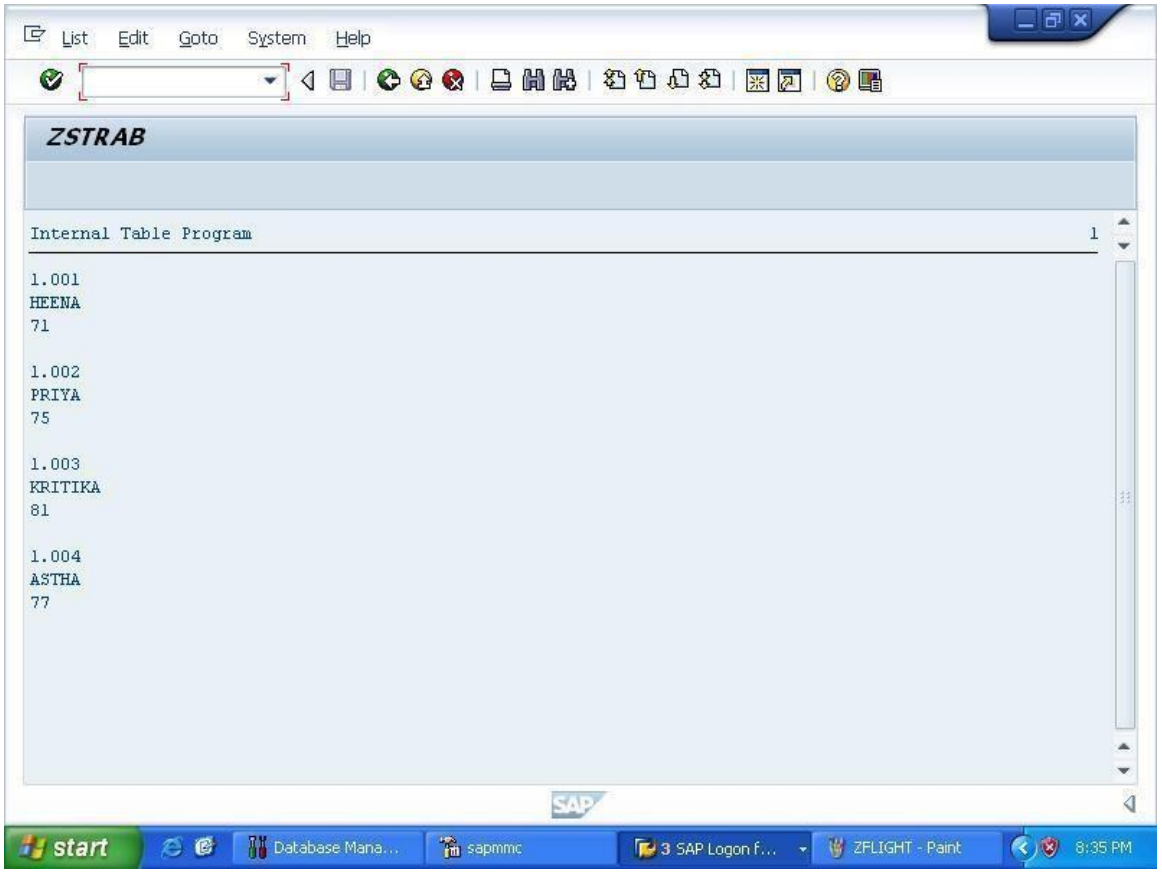
**STUD2-MARKS ='74.50'.

*WRITE : / STUD2-ROLL_NO LEFT-JUSTIFIED, / STUD2-NAME LEFT-JUSTIFIED, / STUD2-MARKS LEFT-JUSTIFIED.

**MOVE-CORRESPONDING STUD2 TO TEACH.

*WRITE: / TEACH-ROLL_NO, / TEACH-ID.

OUTPUT



9.25 WRITE AN EXECUTABLE PROGRAM THAT CONCATENATES TWO WORDS AND WRITE THE RESULT.

&---

*& Report ZSTRINGCAT

*&

&---

*&

*&

&---

REPORT ZSTRINGCAT.

DATA: STR1 TYPE STRING VALUE 'CHANDNI',

STR2 LIKE STR1 VALUE ' KAUNDILYA',

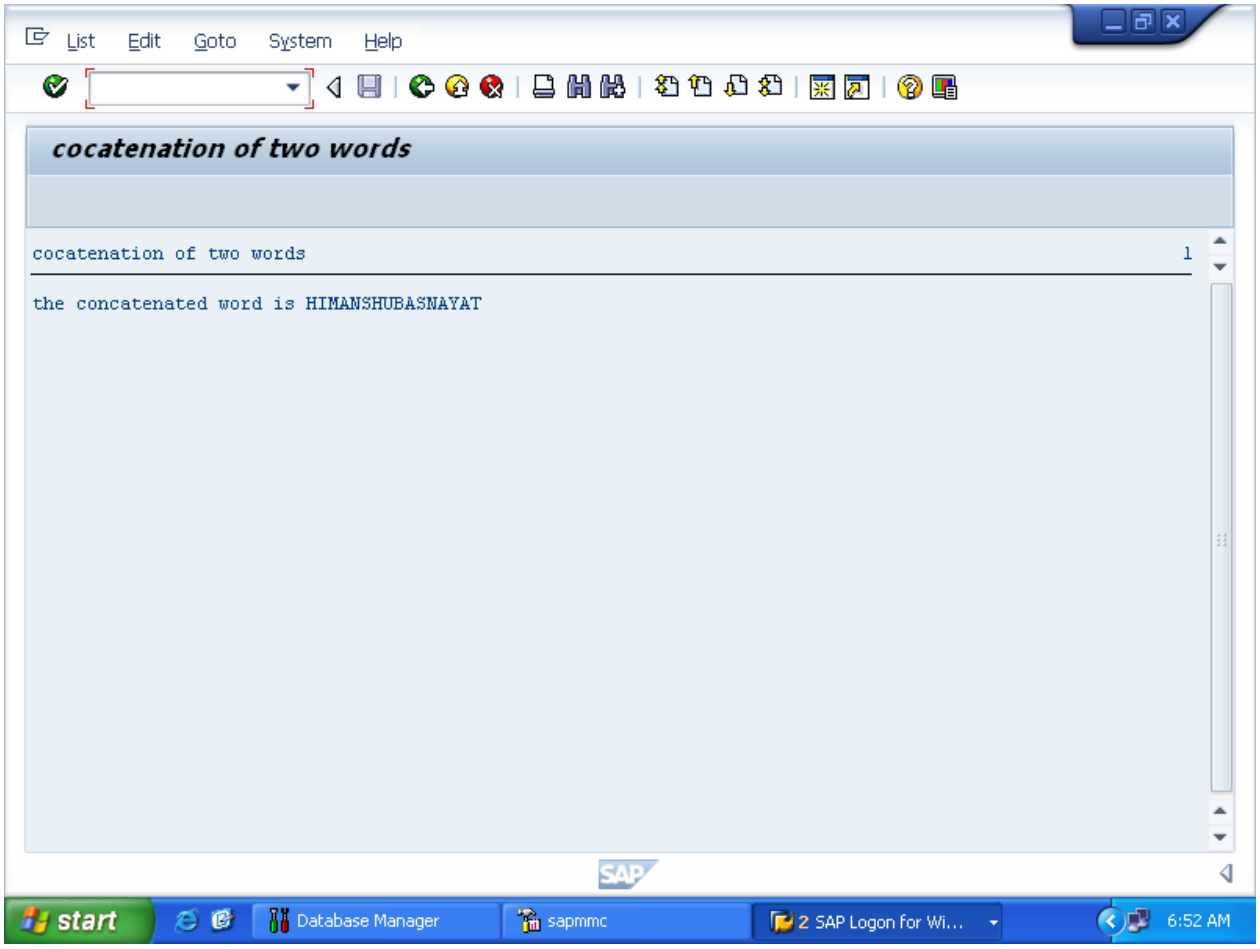
STR3 LIKE STR2.

CONCATENATE STR1 STR2 INTO STR3.

WRITE: 'Concatenated result of ', STR1, ' and ', STR2,

' is ', STR3.

OUTPUT



9.26 WRITE AN EXECUTABLE PROGRAM THAT CREATES A TRANSPARENT TABLE BY PROPAGATING DATA FROM OTHER DATABASE TABLE AND IMPLEMENTS SQL QUERY TO READ RECORDS.

&---

*& Report ZTABLE2

*&

```
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT ZTABLE2.
DATA: it_flight TYPE ZNEW_TABLE.
DATA: wa_sflight TYPE SFLIGHT,
ws_sflight TYPE SFLIGHT.
WRITE:/'Database'.
SELECT * FROM SFLIGHT
INTO wa_sflight
WHERE carrid = 'JL'.
WRITE:/ wa_sflight-carrid,
wa_sflight-connid,
wa_sflight-fldate,
wa_sflight-price.
ENDSELECT.
NEW-LINE.
ULINE.
SELECT * FROM SFLIGHT
INTO TABLE it_flight
WHERE carrid = 'JL'.
WRITE: 'Sorted Internal Table'.
LOOP AT it_flight INTO ws_sflight.
WRITE:/ ws_sflight-carrid,
ws_sflight-connid,
```

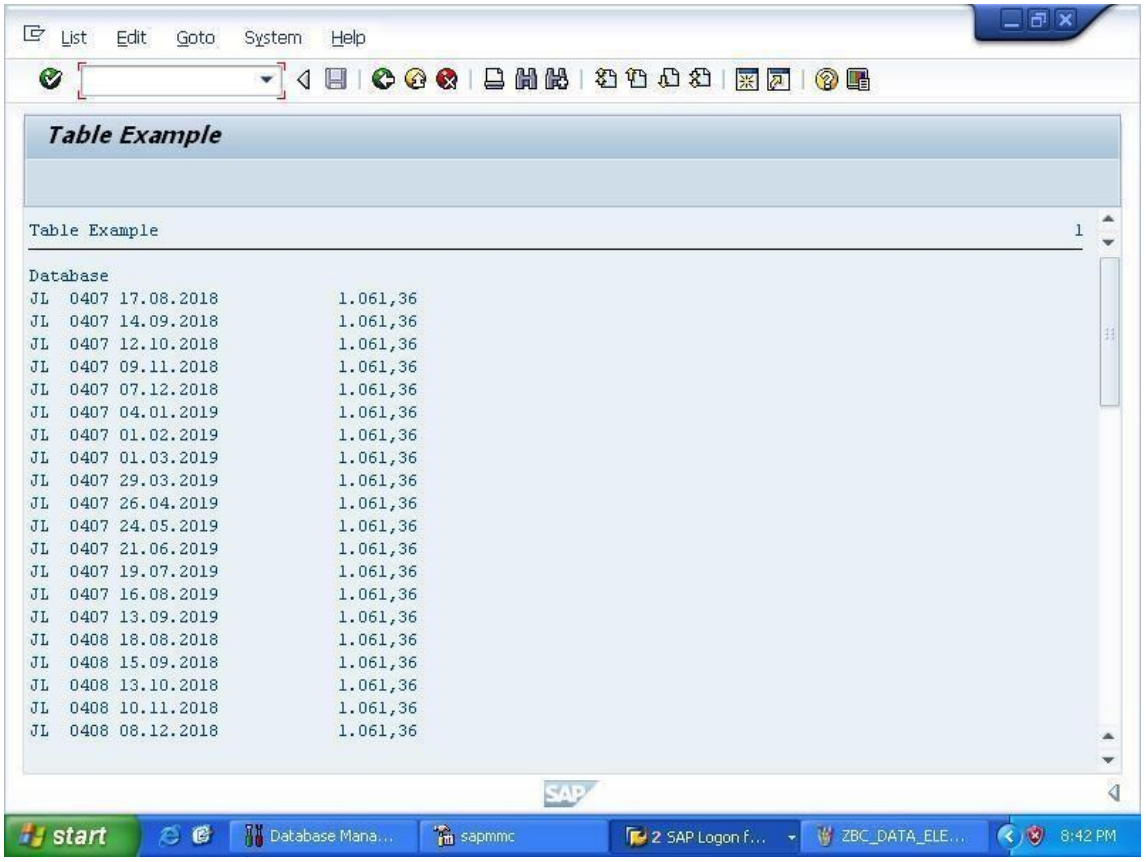
ws_sflight-fldate,

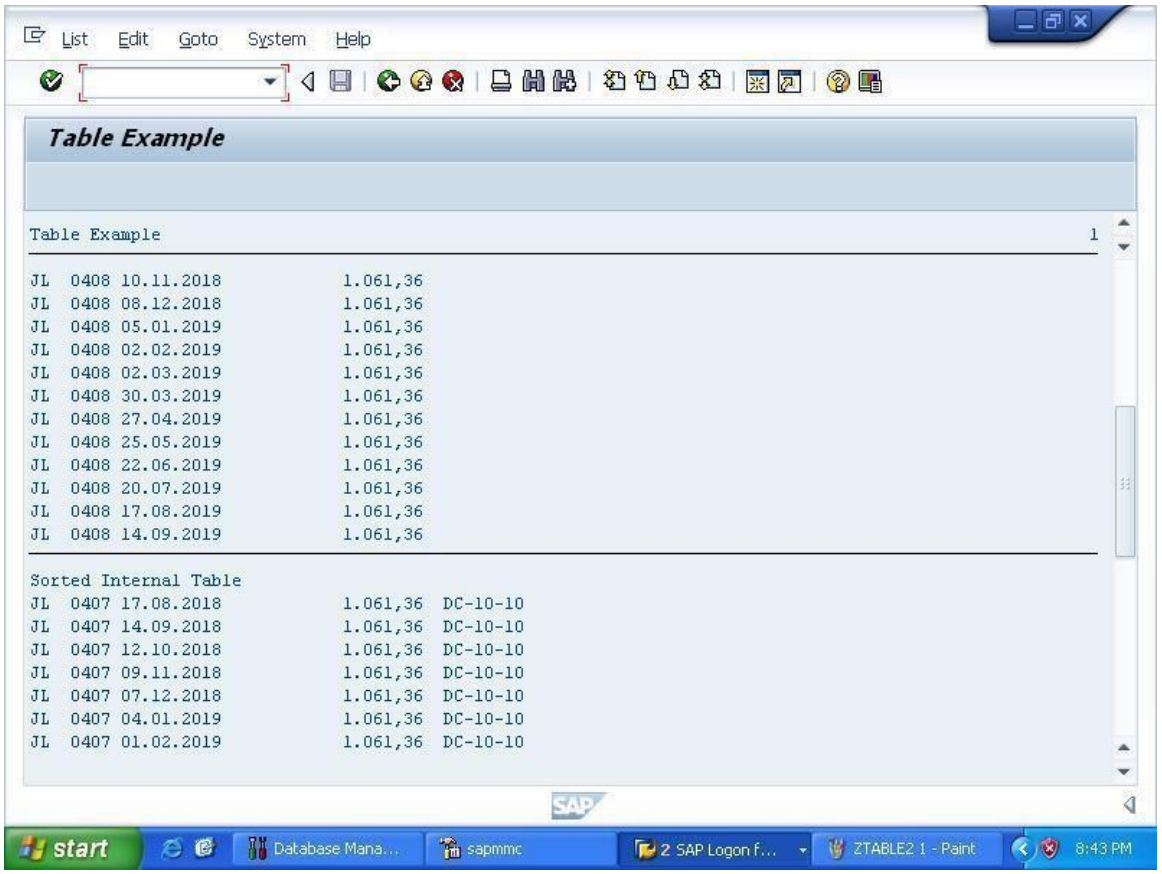
ws_sflight-price,

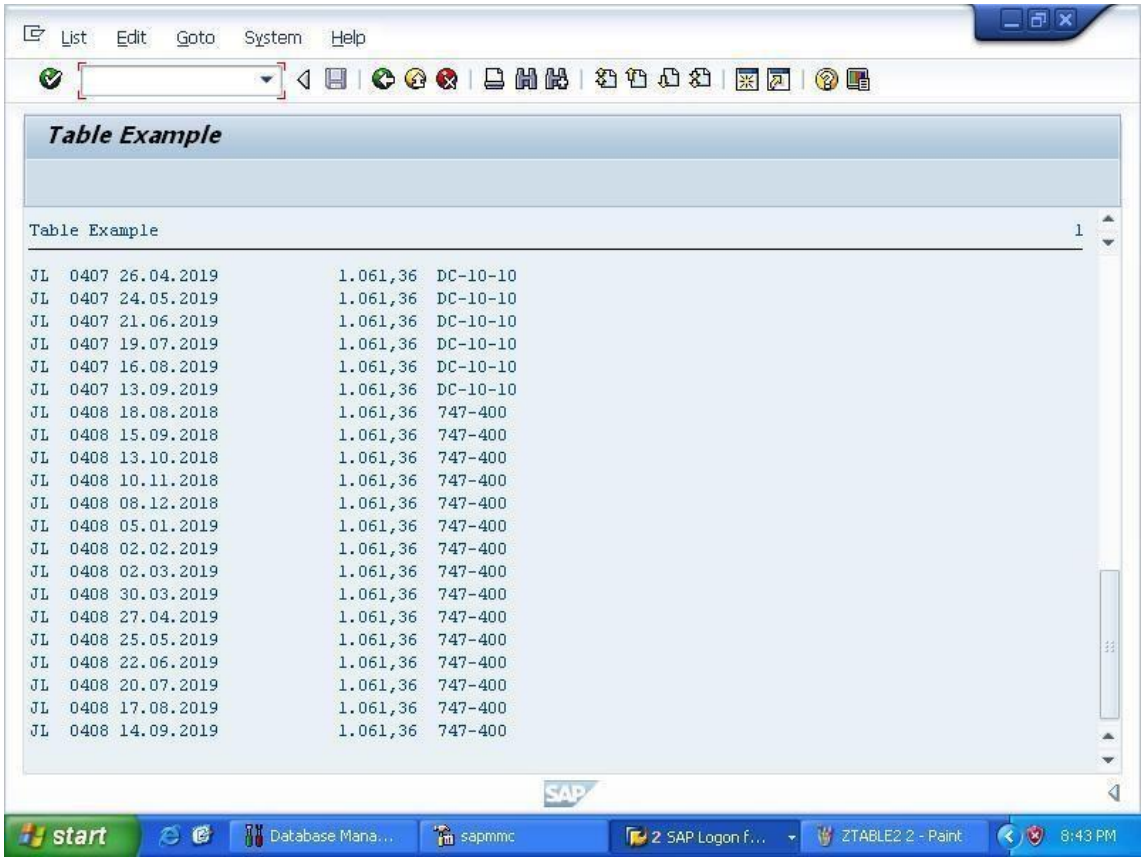
ws_sflight-planetype.

ENDLOOP.

OUTPUT




List Edit Goto System Help
Table Example
Table Example 1
JL 0408 10.11.2018 1.061,36
JL 0408 08.12.2018 1.061,36
JL 0408 05.01.2019 1.061,36
JL 0408 02.02.2019 1.061,36
JL 0408 02.03.2019 1.061,36
JL 0408 30.03.2019 1.061,36
JL 0408 27.04.2019 1.061,36
JL 0408 25.05.2019 1.061,36
JL 0408 22.06.2019 1.061,36
JL 0408 20.07.2019 1.061,36
JL 0408 17.08.2019 1.061,36
JL 0408 14.09.2019 1.061,36
Sorted Internal Table
JL 0407 17.08.2018 1.061,36 DC-10-10
JL 0407 14.09.2018 1.061,36 DC-10-10
JL 0407 12.10.2018 1.061,36 DC-10-10
JL 0407 09.11.2018 1.061,36 DC-10-10
JL 0407 07.12.2018 1.061,36 DC-10-10
JL 0407 04.01.2019 1.061,36 DC-10-10
JL 0407 01.02.2019 1.061,36 DC-10-10
start Database Mana... sapmmc 2 SAP Logon f... ZTABLE2 1 - Paint 8:43 PM


List Edit Goto System Help
Table Example
Table Example 1
JL 0407 26.04.2019 1.061,36 DC-10-10
JL 0407 24.05.2019 1.061,36 DC-10-10
JL 0407 21.06.2019 1.061,36 DC-10-10
JL 0407 19.07.2019 1.061,36 DC-10-10
JL 0407 16.08.2019 1.061,36 DC-10-10
JL 0407 13.09.2019 1.061,36 DC-10-10
JL 0408 18.08.2018 1.061,36 747-400
JL 0408 15.09.2018 1.061,36 747-400
JL 0408 13.10.2018 1.061,36 747-400
JL 0408 10.11.2018 1.061,36 747-400
JL 0408 08.12.2018 1.061,36 747-400
JL 0408 05.01.2019 1.061,36 747-400
JL 0408 02.02.2019 1.061,36 747-400
JL 0408 02.03.2019 1.061,36 747-400
JL 0408 30.03.2019 1.061,36 747-400
JL 0408 27.04.2019 1.061,36 747-400
JL 0408 25.05.2019 1.061,36 747-400
JL 0408 22.06.2019 1.061,36 747-400
JL 0408 20.07.2019 1.061,36 747-400
JL 0408 17.08.2019 1.061,36 747-400
JL 0408 14.09.2019 1.061,36 747-400
start Database Mana... sapmmc 2 SAP Logon f... ZTABLE2 2 - Paint 8:43 PM

CONCLUSION

ABAP (Advanced Business Application Programming) is a programming language for developing applications for the SAP R/3 system, a widely-installed business application subsystem. The latest version, ABAP Objects, is object-oriented programming. SAP will run applications written using ABAP/4, the earlier ABAP version, as well as applications using ABAP Objects.

SAP's original business model for R/3 was developed before the idea of an object-oriented model was widespread. The transition to the object-oriented model reflects an increased customer demand for it. ABAP Objects uses a single inheritance model and full support for object features such as encapsulation, polymorphism, and persistence.

At the end of the training, we experienced and learned how an ERP system under SAP works and how we can program it with ABAP.

Under ABAP we learnt the following concepts

1. Reports
2. Module Pool Programming
3. Interfaces
4. Forms
5. Data conversions
6. User Exits & BADI

REFERENCES

1. TAW10 ABAP Workbench Fundamentals – Part 1 by SAP INDIA.
2. TAW10 ABAP Workbench Fundamentals – Part2 by SAP INDIA.
3. TAW12 ABAP Workbench Concepts – Part1 by SAP INDIA.
4. TAW12 ABAP Workbench Concepts – Part2 by SAP INDIA.
5. Beginners Guide to SAP ABAP by Peter Moxon.
6. Sap ABAP/4 by Kogent Learning Solutions Inc.
7. Concepts from www.tutorialspoint.com.