**SAP OOABAP**

**A screenshot of a phone

AI-generated content may be incorrect.**

**Object Oriented ABAP**

**What is Object Orientation?**

In the past, Information System used to be defined primarily by their functionality: Data and Functions were kept separate and linked together by means of input and output relations.

The Object-Oriented Approach focuses on Objects that represent abstract or concrete things of the real world. These Objects are defined by their character and their properties, which are represented by their internal structure and their attribute (data). The behavior of these objects is described by the methods (functionality).

**Comparison between Procedural and Object Oriented Programming:**

* **Emphasis:**
  + Procedural Programming, Emphasis is on task
  + Object Oriented Programming ,Emphasis is on things that does those tasks.
* **Modularization:**
  + Procedural Programming: Programs are divided into small programs 🡪 Function
  + Object Oriented Programming: Programs are organized into classes and objects and the functionalities are embedded into methods of a class.
* **Data Security:**
  + Procedural Programming: Most of the functions shares Global Data.
  + Object Oriented Programming: Data can be hidden and cannot be accessed by external sources.
* **Extensibility:**
  + Procedural Programming: Relatively more time consuming to modify any existing functionality.
  + Object Oriented Programming: New data and Functionalities can be easily added whenever necessary.

**Object Oriented Approach – Key Features**

* Better Programming Structure.
* Real world entity can be easily modeled.
* Stress on data security and data access.
* Reduction in Code Redundancy.
* Data Encapsulation and abstraction.

**What are Objects and Classes?**

**Class:** An Object is a section of a source code that contains data and provided services. The data forms the attributes of an object and the Services are known as Methods (Operations or Functions). They form a capsule which combines the character to the respective behavior. Objects should enable programmers to map a real problem and its proposed software solution on a One-to-One basis.

**Object:** Classes describe objects. From a technical point of view, Objects are the runtime instance of a class. In theory, we can create any number of objects based on a single class. Each instance (object) of a class has a unique identity and its own set of values for its attributes.

**Local and Global Class:** Class is an abstract description of an object. Class in ABAP Objects can be declared either locally or globally.

**Global Class:** Global Classes and Interfaces are defined in the Class Builder (Transaction Code SE24) in the ABAP Workbench. They are stored centrally in the class pools in the class library in the R/3 repository. All of the ABAP Programs in an R/3 system can access the global classes.

**Local Class:** Local Class are define in an ABAP Program (Transaction SE38) and can only be used in the program in which they are defined. Every class will have 2 Sections:

* **Definition:** This section is used to declare the components of the classes such as attributes, methods, events. They are enclosed in the ABAP Statement CLASS ….. ENDCLASS.

CLASS <name> DEFINITION.

ENDCLASS.

* **Implementation:** This section of a class contains the implementation of all Methods of the class. The implementation part of a local class is a processing block.

CLASS <name> IMPLEMENTATION.

ENDCLASS.

**Comparison between Local and Global Class:**

* **Accessed By:**
  + Global Class can be accesses by any program.
  + Local Class can only be accessed by the program inside which it has been defined.
* **Stored In:**
  + Global Class are stored in the Class Repository.
  + Local Class are stored in the program where it has been defined.
* **Created By:**
  + Global Class are created using Transaction SE24.
  + Local Class are created using Transaction SE38.
* **Namespace:**
  + Global Class must begin with either “Z” or “Y”.
  + Local Class can begin with any character.

**Structure of a Class:**

* A Class contains components
* Each component is assigned to a visibility section.
* Classes implement methods.

**Components of a Class:**

* **Attributes:** Any data, constants, types declared within a class form the attributes of the class.
* **Methods:** Block of code, providing some functionality offered by the class. Can be compared to Function Modules. They can access all of the attributes of a class. Methods are defined in the DEFINITION part of a class and implement in the IMPLEMENTATION part using the below processing block:

METHOD <name>.

………….

ENDMETHOD.

Methods are called using the CALL METHOD Statement.

* **Events:** A mechanism set within a class which can help a class to trigger methods of another/other class.
* **Interfaces:** Interfaces are independent structures that we can implement in a class to extend the scope of that class.

**Visibility of Components:**

Each Class component has a visibility. In ABAP Objects, the whole class definition is separated into three visibility sections:

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

CLASS <name> DEFINITION.

PUBLIC SECTION.

. . . . .

PROTECTED SECTION.

. . . . .

PRIVATE SECTION.

. . . . .

ENDCLASS.

**Assignment for OO ABAP 1:** Create a program for arithmetic calculation from Object Oriented Approach. You will have 4 different methods for Addition, Subtraction, Multiplication and Division. Based on the radio button selection call the methods accordingly. Take 2 constants (lc\_num1 value 100 and lc\_num2 value 50) and 1 variable (lv\_result) for the calculation.

**Assignment for OO ABAP 2:** Do the same program by using Parameters in the local class. Example given below

METHODS: add importing

num1 type i

num2 type i

exporting result type i.

CALL METHOD obj->add

EXPORTING

num1 = p\_num1

num2 = p\_num2

Importing

result = res.

**Assignment for OO ABAP 3:** Create a global class that will keep a track of your bank account. Then write a program to use this class. Below are the details:

* Class Name: ZCLACCOUNTxx 🡪 xx is your ID 🡪 if you id is 2 🡪 ZCLACCOUNT02
* Public Methods:
  + SET\_BALANC (sets the balance to a new value)
    - Import Parameter
      * NEW\_BALANCE type DMBTR
  + DEPOSIT (adds a deposit amount to the balance and returns the new balance)
    - Import Parameter
      * AMOUNT type DMBTR
    - Export Parameter
      * NEW\_BALANCE
  + WITHDRAW (subtracts a deposit amount from the balance and returns the new balance)
    - Import Parameter
      * AMOUNT type DMBTR
    - Export Parameter
      * NEW\_BALANCE type DMBTR
    - Exception
      * INSUFFICIENT\_FUNDS
* Selection Screen
  + Parameters:
    - P\_AMNT type DMBTR 🡪 set the balance (4000)
    - P\_DPST 🡪 Deposit the amount (3000)
    - P\_WDRW 🡪 Withdraw (200)
  + Output
    - Set Balance to 4,000.
    - Deposited 3,000 making New Balance as 7,000
    - Withdrew 200 making New Balance as 6,800
    - If we try to withdrew more than the actual balance we will get the Exception as Insufficient Funds

**Assignment for OO ABAP 4:** Create a global class which will display the Purchase and Sales Document related information as an Output. Appended are the details:

* Selection Screen
  + Radio Buttons
    - Purchase Information
      * Select Options
        + Purchasing Document
    - Sales Information
      * Select Options
        + Sales Document
* Based on user input extract Purchase Information by Joining EKKO and EKBE tables
* Based on user input extract Sales Information by Joining VBAK and VBAP tables.
* Global Class Design
  + Class Name 🡪 ZCLTABLExx 🡪 xx will be your ID
  + Methods
    - GET\_PURCH and GET\_SALES
* Display the result in ALV output

**Instance Attributes / Components:**

Instance attr exist separately in each instance (object) of the class and are referred using instance component selector **‘->’**. The instance attributes / components can be defined using normal ABAP Statements such as DATA, METHODS etc.

**Static Attributes / Components:**

Static components only exists once per class and are valid for all the instances of the class. They are declared with the **CLASS-** Keyword. Static components can be used without even creating an instance of the class and are referred to using static component selector **‘=>’**.

**Attributes of Object Oriented Programming**

* **Inheritance:** Inheritance is the concept of adopting features from the parent and reusing them. It involves passing the behavior of a class to another class. You can use an existing class to derive a new class. Derived classes inherits the data and methods of the super class (parent class). However, they can overwrite existing methods and also add new ones.
  + **Types of Inheritance:**
  + **Single Inheritance:** Acquiring properties from Single parent.

PARENT CLASS

CHILD CLASS

* + - **Multi-level Inheritance:** Acquiring properties from multiple parents in terms of level / hierarchy.

PARENT CLASS

CHILD CLASS Level 2

CHILD CLASS Level 1

* + - **Multiple Inheritance:** Acquiring properties from multiple parents

PARENT CLASS

PARENT CLASS

PARENT CLASS

CHILD CLASS

Syntax: CLASS parent DEFINITION.

…………

ENDCLASS.

CLASS child DEFINITION INHERITING FROM parent.

…………

ENDCLASS.

**Assignment for OO ABAP 5:** Create a program with following details:

* Classes:
  + Parent Class: lcl\_inst\_teacher
    - Methods:
      * constructor 🡪 increment the total number of teachers by 1
      * add\_teacher 🡪 add a new teacher
      * display\_teacher 🡪 displays all the teachers
      * display\_total\_teacher 🡪 displays total number of teachers
  + Child Class:
    - lcl\_com\_teach
      * Methods
        + Constructor with importing parameters
        + Add\_teacher 🡪 redefine the super class method here by calculating the salary of a teacher based on total number of days \* 8000.
    - lcl\_art\_teach 🡪 same as above
    - lcl\_scn\_teach 🡪 same as above
* Display the list of Arts, Commerce and Science Teachers separately with their salary in an ALV Format.
* Selection Screen
  + Radio Button
    - Commerce Teachers Information
    - Science Teachers Information
    - Art Teachers Information
* Add at least 10 teachers information for each department (science / art / commerce)
* Display only the details of those teacher whose radio button has been selected.
* Fields to be displayed: Teacher ID, First Name, Last Name, Salary, DOB and Contact.

**Assignment for OO ABAP 6:** Create a program with following details

* Selection Screen
  + Radio Button
    - Material Information
      * Select Options
        + MATNR
    - Sales Information
      * Select Options
        + VBELN
    - HR Information
      * Select Options
        + PERNR
* Radio Buttons will be present in 1 block and Select options will exist in another block.
* Create a local class to extract the information based on the radio button selected and the respective values inputted by user in SELECT-OPTIONS.
* Create 2 Custom Containers and display the outputs from 2 different tables for each radio button selection.
  + If Material is selected then
    - Custom Container 1 will show output from MARA
    - Custom Container 2 will show output from MARC
  + If Sales is selected then
    - Custom Container 1 will show output from VBAK
    - Custom Container 2 will show output from VBAP
  + If HR Information is selected then
    - Custom Container 1 will show output from PA0002
    - Custom Container 2 will show output from PA0001
  + Display 10 Fields from each table.
* **Abstraction:** Everything is visualized in terms of classes and objects.
* **Encapsulation:** The wrapping up of data and methods into a single unit (called Class) is known as Encapsulation. The data is not accessible to the outside world only those methods, which are wrapped in the class can access it.
* **Polymorphism:** Methods of same name behave differently in different classes (Identically names methods behave differently in different classes).

**Super:** Super is a key word used to represent the super class of a class. In OOPS you can access the methods and attributes of the super class in child class using the word SUPER.

**Assignment for OO ABAP 7:** Create a program with following details

* Selection Screen
  + Radio Button
    - Material Information
      * Parameter
        + MATNR
    - Purchase Information
      * Parameter
        + EBELN
    - HR Information
      * Parameter
        + PERNR
* Radio Buttons will be present in 1 block and Select options will exist in another block.
* Create a Global class to extract the information based on the radio button selected and the respective values inputted by user in PARAMETER.
  + Parent Class will only give you details for MARA, EKKO and PA0000.
  + Sub Class will redefine all these methods and will give you the above data as well as data from MARC, EKBE and PA0002.
* Create 2 Custom Containers and display the outputs from 2 different tables for each radio button selection.
  + If Material is selected then
    - Custom Container 1 will show output from MARA
    - Custom Container 2 will show output from MARC
  + If Purchase is selected then
    - Custom Container 1 will show output from VBAK
    - Custom Container 2 will show output from VBAP
  + If HR Information is selected then
    - Custom Container 1 will show output from PA0000
    - Custom Container 2 will show output from PA0002
  + Display 10 Fields from each table.

**--------------------------------------------------------------------------------------------------------------------**

SAP OOABAP – Object oriented ABAP – Used in HANA.

OOABAP (Object-Oriented ABAP) is an extension of traditional **ABAP** that allows for object-oriented programming (OOP) concepts like **classes, objects, inheritance, polymorphism, and encapsulation**. It is used in modern SAP applications, especially with **ABAP on HANA, Web Dynpro, and SAP Fiori**.

Due to the limitation in the events/some disadvantage – we are using sap OOABAP.

OOPS – **Design the class /methods based on the requirements.**

Object maps to the real time entity.

Used in complex,large and frequently updated.

Difference b/w procedural abap and ooabap:

**Class** -> a collections of data/function/methods etc..

* Local Class – scope within the prog and can’t access globally or by any prog / whenever we want. Create in se38.
* Global class – Access globally by any other prog and scope is global – se24.
* Class composed of attributes – characteristics of class & methods.

Global Class types:

1.Usual ABAP Class – purpose to write the logic(FM).

2.Exceptional class. – Purpose to raise and handle exception.

3.Persistance class – Purpose to perform db operations(insert,delete & update)

4.Unit test class – write unit test cases.

* **Final class** means it is class which cannot be inherited – does not have subclass.
* **Abstract class :**

**Object** -> Instance of the class.

* To access / use the class properities , we need to create the object whatever.

**Concepts of OOPS:**

1.Inheritance : (Re-usablity)

Derive new class from the existing class or parent class or super class and it can use the data , methods etc of the parent class.

2.Encapsulation :

The wrapping up of data and methods into a single unit (called Class) is known as Encapsulation. The data is not accessible to the outside world only those methods, which are wrapped in the class can access it.

3.Polymorphism : (Same Method, Different Behavior) – method overriding – redefines.

Child class redefines the parent method & implement the different logic for that method. Same method with different functionalities in the different classes.

4.Abstraction

**Properties of Obj.oriented:**

Local Class definition and implementation

--------------

Class <class\_name> **definition**.

**{Visibility Section**} - Public/Protected/Private

{**Declarations - Attribute/Method**} - Static/Instance method Block

**{Events/Interface}**

Endclass.

-------------------

Class\_name can be anything.

Class <class\_name> **implementation.**

Method <method\_name>.

------------------ Processing block

Endmethod

Endclass.

In Implementation , Can write abap code like get data, manipulate it and so on. Not defining anything like variable,methods etc..

Class Implementation / processing block should be end of the program. No executable statement after implementation.

Create an obj before implementation or If needed, we should use “**start of selection**” event and then use executable statement like create an obj for the class.

**Object:**

Create object – syntax is given below :

1. Data **obj\_class** type ref to <class\_name> - declare a obj var.

Create object obj\_class.-> Create an object.

1. Data (**obj\_class**) = new <class\_name>(). “another way ti create a obj for a class – new syntax.

obj\_class –> give the user defined obj name.

How to access the variable and method outside of the class(public only).

* Obj\_name **->** variable name or method name – when we are calling by obj of a class.
* Class\_name **=>** variable name or method name – when we are calling by class name.
* We can access the protected and private attr / methods inside the class ( **Implementation part la**).

**Class components:**

**1.Visibility Section:**

Public – declared some x – can be accessed outside the class/within the class / in subclass.

Private - declared some x – cannot be accessed outside the class and accessed within the same class.

Protected - can be accessed within the class or in subclass.

**2.Methods :**

**A screenshot of a computer

AI-generated content may be incorrect.**

Instance Method : -> Every obj has its own memory – no common memory space. So change in ins\_attr does not reflect in other obj of the class as well.

* + - Can be accessed by an obj.
    - Inside this method, we can call other static and instance method & also access static and instance variable.
    - Declared by using the **METHODS** statement.
    - Need to use “->” to call the method/access the attr by the obj name.

Static Method : - it is commonly shared by all the obj of the class – shared/one memory space . Change in static\_attributes reflect in all other obj of the class.

* can be accessed by class or by obj.
* Need to use “=>” to call the method/access the attr by the class name.
* Inside this method, we can call only other static method and only static.attr can be accessed.
* Declared by using the **CLASS-** **METHODS** statement

So in SE24 , if we declare both static and instance methods, when we execute the class it shows only the static method becoz it does not need obj to call the method.

To call the instance method , we need to create obj in the se24 even for testing and check it.

**STATIC , INSTANCE ATTRIBUTE:**

**Depend on the requirement -> we need to decide the static or instance attr/method.**

To define the method inside the method, we need to use ‘**me**’ keyword - reference to the current instance of a class within a method of that class .

**Method**

Eg : me -> method\_name().

**Endmethod.**

**Static/Global: no need to create obj – can be accessed using obj and classname.**

* It is globally linked to the class – single copy for a class (exist once per class) in memory – have common/same memory for every obj of an class which mean a change a static attribute in one object of a class, the change is reflected/visible to all other objects of the class as well.
* Scope of the attribute – single/one object of the class
* Need to use “=>” to call the method/access the attr by the class name.
* Static.Attribute :
  + Accessed by class or by obj .
  + Can be accessed by static or instance method.
  + A static attribute is declared by using the **CLASS-DATA** statement.

**Instance/Local**: multiple copies of an space for each obj – need to create obj to call this obj – not accessed by class-name.

* Every obj has its own memory – which means a change a instance attribute in one object of a class, the change is not reflected/visible to all other objects of the class as well.
* Scope of the attribute – every/multiple object of the class.
* Me-> - this keyword is used when we calling instance/static method inside the instance method.
* Me-> this keyword is also used when we declare a same local variable name inside a method which is already defined globally for that class. To call global variable name, we use me->varible\_name

Eg:

A screenshot of a computer

AI-generated content may be incorrect.

* Instance.Attr :
  + Accessed only by obj of a class.
  + Can be accessed only in the instance method.
  + A static attribute is declared by using the **DATA** statement.

Method Parameter:

Importing <= getting/giving a data to the method.

* More than one imp. parameter/var.
* Call/Pass by reference or call by value.

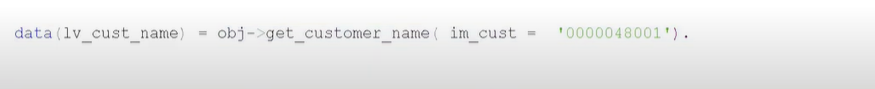
Exporting => sending the data/info back.

* More than one exp.parameter/var.
* Call/Pass by reference or call by value.

Changing ⬄ we will receive the data and send back the updated data again to the prog/where we call the method.

* More than one changing parameter/var.
* Call /Pass by reference or call by value.

Returning => send the data/info back.

* Return only one value/parameter.
* Call/Pass by value.
* When the method has a return parameter, we can call the method like below because anyways it will return single value only.
* 

Pass by value - it carries the value of the variable from main program to subroutines/FM, so if any changes happens to variable anywhere it won't affect the source value.

Pass By Reference - it carries the address of the variable from main program to subroutines/FM, so if any changes happens anywhere to the variable then it affects the source and also wherever this variable has been used.(Global).

Syntax:

Methods display importing value(x) type y

Exporting value(x) type y

Changing value(x) type y

Returning value(x) type y.

Constructor :

* Special type of method which is execute automatically when an object is created.
* Used to initialize the value for the attribute in the class.
* Can’t call using class or obj name or using call method.
* Have predefined name.
* Won’t return any parameter.
* If there is spelling mistake , it will take it as a normal method
* One constructor taan per class.

**Types:**

1. Instance Constructor :

* Method name : methods ‘CONSTRUCTOR’.
* Keep it in public visibility.
* Has importing parameter & exception , no exporting parameter.
* Access both static & instance attr.
* Used to set default values for a particular instance/obj.
* Called everytime when an obj is created.

1. Static Constructor : -A **static constructor** is a special method that runs **only once per class**—no matter how many objects you create.

* Method name : class-methods ‘CLASS\_CONSTRUCTOR’.
* Single static constructor for a class.
* It executes **automatically** when the class is accessed for the first time.
* It is used to **initialize static attributes**.
* if another object is created later, the **static constructor will NOT execute again**.
* No parameters are allowed.
* Visibility : Public.

Inheritance :

Derive new class from the existing class or parent class or super class .

It inherits/use the data , methods etc of the parent class.

Able to overwrite the parent class method/data.

Generally to use any super class methods with diff.functionalities in child class, we need to redefine it In the subclass.

To use the super class method as it is in child class, directly it call from obj of the child class

Adv : Reusablity , include additional concepts without overwrite the base class functionality.

Disadv : In ofter , some of the parent class attr/prop left unused in the derived class – leads to mem.wastage , base class change might affect all the derived class , slow processing.

**Properties :**

**Super class:**

* + Super class – it is the parent class where it passes the prop to the derived class.
  + Not a final class – **Final class** means it is class which cannot be inherited – does not have subclass.
  + Visibility – Protected & public sections – accessed in the subclass . Can’t access private section.(Friend class)
  + Constructor.
  + **Super Keyword** -> used in inheritance -only to call super class methods not attr- helps to access/call methods of the parent class from sub-class to overridden it.

The method must be redefined to use the super keyword and Used inside the same method name being redefined ( **SUPER->display( ) inside display( )** )

It is mainly used in:

**Constructor Methods** (to initialize the parent class).

**Overridden Methods** (to reuse parent logic and extend it)

Subclass:

* Subclass can also be a super class.
* Subclass can use super class attributes by mentioned its attr\_name. Overwrite it by assigning new value to the super class attr.
* To call the method , we need super keyword.
* Constructor.
* Redefinition -> used for methods – inherited the methods from parent class to sub-class - redefine the inherited method of parent class in sub-class to add our own logic in that method.

Inheritance with constructor:

* Super and sub class constructor optional.
* Subclass constructor can have constructor without constructor in super class.
* Parent constructors do NOT execute automatically in child classes.
* Use SUPER->constructor( ) to call the parent constructor explicitly.
* If the parent constructor has parameters, the child must pass them explicitly.
* SAP ABAP does not support constructor overloading, but default parameters can mimic it.

Interface:

It is kind of template/independent structure which defines methods and attributes inside based on our requirement. Interface can only contain empty method declaration and components .Implementation can be done through the class not having its own implementation.

* Used in multiple class.
* **Need to implement all the methods defined in the interface in the class we are using that interface else it throws an error.**
* Same interface methods/attr can be used in diff.class with different functionalities.
* Re-useability and maintain structure.
* Methods are in public visibility section by default.
* The class **implements** the interface using **INTERFACES zif\_vehicle(interface\_name)**.
* Methods are implemented using **interface\_name~method\_name** format to specify the interface method

**Abstract method:**

Abstract method is that method which has only definition in the base class.

There will not be any Implementation of abstract method in the abstract class.

We can create objects of Abstract class through sub class of those abstract class.

**Abstract class:**

It is a specical type of class which has **at least one abstract method**.

We cannot create object of Abstract class.

We can create objects of Abstract class through sub class of those abstract class.

Can have instance, static and abstract methods.

A **subclass must implement all abstract methods**, or it will also be **abstract**.

**.**

Difference b/w Abstract & Interface:

* **Multiple Inheritance:** We can achieve multiple inheritance using Interfaces. Since ABAP doesn’t support more than one Super class, we can have only one abstract class as Super class.
* **New Functionality**: If we add a new method in the Interface, all the implementing classes have to implement this method. If we don’t implement the method, it would result into Run-time error. For Abstract class, if we add a non-abstract method, its not required to redefine that in each and every inherited class.
* **Default Behavior**: We can implement the non-abstract method in abstract class. We can’t have any implementation in Interface as it only contains the empty method definition.
* **Visibility:** All interface components are PUBLIC by default. For Abstract class, we can set the visibility of each component.
* **Purpose :** Abstract class can be used if we need common code/functionality/implemention + abstract methodsas a part of inheritance whereas interface can be used if empty definitions are to be used in different inherited class with the own logic.

Note:

**Multiple inheritance is not possible in SAP OOABAP but can be achieved using defining multiple interface and used in one class which is called as super class/parent class then we inherit from that super class.**

**Instance (concrete) methods of an abstract class are inherited normally and can be used without redefinition.**

**Only abstract methods must be redefined in the subclass.  
Concrete methods can be redefined if we want to change their behavior**

Events:

In SAP OOABAP (Object-Oriented ABAP), **events** are used to implement the **Publisher-Subscriber** design pattern, where one class (publisher) triggers an event, and other classes (subscribers) respond to it.

An event in OOABAP is a mechanism where one class (publisher) triggers an event, and another class (subscriber) listens and responds to it. It helps in communication between objects without direct dependency."

Event is an action or occurrence of something that is recognized by the application. Handling events means response by the application to the action or that occurrence.

Events are the message raised by an object. Whenever any condition is met, object can raise an event. E.g. Timer has expired, Cut-off Limit has reached, User Interaction happened etc. This event than can be caught by the receiver. The receiver than implements the logic to handle the event. E.g. When Timer has expired, render the page again; When cut-off limit is reached, tell user to stop etc.

**Types of Events in OOABAP:**

**Instance Events** – Triggered by an instance (object) of a class.

**Static Events** – Triggered at the class level (not tied to an object).

How event Works:

Define an Event in a class.(EVENTS)

Trigger (Raise) the Event based on some conditions.(RAISE EVENT)

Handle the Event in another class using a method.

Register and Trigger the Event(Set\_Handler)

Ex : Let’s assume we have a scenario where a **Car** class has an event **speed\_limit\_crossed** that triggers when the speed crosses 100 km/h. Another class **TrafficControl** listens to this event and gives a warning. (or)

Doorbell rang(Event & event triggers) and houseowner will listen and respond to it.(Handle the event)

Need to study :

**OOALV(Important)**

Design patterns.