Discussion Topics

- Class Definition
  - Attributes
  - Methods
- Simple Object Instantiation
Object Oriented ABAP Development

OO often considered superior to procedural programming due to encapsulation of data with related functionality.
ABAP Objects similar conceptually to languages like C++ and Java.
Backwards compatible with procedural ABAP, so objects can be used in procedural coding.
Some language constructs considered deprecated (yet still supported in procedural ABAP) are disallowed in ABAP objects.
Just as in other OO languages, ABAP Objects supports:

- **Inheritance**—one class shares structure and behavior with another
- **Polymorphism**—different (but related) objects have the same communication interface
- **Events**—objects respond when triggered. Objects can trigger an event.

OO ABAP

**Class**—definition of the structure and functionality of an object-to-be.
Defines data the object will contain (**attributes**) and related functionality (**methods**).
ABAP class syntax divides the class **definition** (containing attribute declarations and method interface definitions) and the class **implementation** (method functionality definitions).

```abap
CLASS exampleclass DEFINITION.
  ENDCLASS.

CLASS exampleclass IMPLEMENTATION.
  ENDCLASS.
```

Can be defined locally or globally (using **Class Builder** trans SE24).

**Object**—instantiation of a class, storing actual data and/or allowing method execution.
Attribute Definition

A class definition may define the following: types, constants, data objects (attributes), and method interfaces.

Items may be designated **public**—available from outside the object, or **private**—available only within the object itself.

A public attribute may be designated **READ-ONLY**.

Within the class definition, the public section must come before the private section.

```
CLASS exampleclass DEFINITION.
  TYPES "definition of data types
  PUBLIC SECTION. [PRIVATE SECTION.]
  CONSTANTS "definition of constants
  DATA "definition of data objects
  DATA name TYPE type READ-ONLY. "read-only
ENDCLASS.
```

Static Attributes

A **static attribute** is an attribute whose value is shared among all objects in a class and is not dependent on instantiation.

Often used for things like object instance counters.

Since the data belongs to the class and is shared among the objects, it is declared as **CLASS-DATA**.

```
CLASS exampleclass DEFINITION.
  PUBLIC SECTION. [PRIVATE SECTION.]
  CLASS-DATA name TYPE type.
ENDCLASS.
```

Attributes (static or instance) may use **VALUE** upon declaration to set a default value.
Quick Practice

Method Definition

In the **class definition**, the **method definition** indicates a method's **interface**.

- Method parameters
- Method return type (**method signature**)  
- Method exception (to be covered later)

Methods may be designated as **public** or **private**.

A private method may only be called from within another method of the class.

**CLASS-METHODS** may be defined. These methods may only access static attributes.

Class methods may be invoked either using the name of the class or on an object of the class.

An object does not need to be instantiated to invoke a class method.
Method Parameters

Method parameter types:

**IMPORTING**—input parameters
- Used to transfer data to a method. May be designated as OPTIONAL. May have a DEFAULT value specified.
- Value of parameter may not be changed in method definition. (system enforced)

**EXPORTING**—output parameters
- At end of method value of formal parameter copied back to actual.
- Export parameters are **always** optional (by default).
- Intent is for the incoming value of the parameter to not be used (**not** system enforced).

**CHANGING**—combination of above
- Passed to method for use. Can be changed.
- May be designated as OPTIONAL. May have a DEFAULT value specified.
- With any of the above, parameters may be designated pass by value.

Method return values

A method may be defined to return a single value.
This is specified by listing **RETURNING** in the interface.
If **RETURNING** is indicated, **EXPORTING** and **CHANGING** parameters may not be specified.

The return operation is not done with an explicit RETURN statement but rather by assigning a value to the name of the data object listed in the **RETURNING** statement.

A method with a return value is called a **functional method**. Unlike other methods, functional methods they can be called within other statements such as IF, CASE, and WHILE.
Example method definition

CLASS exampleclass DEFINITION.
PUBLIC SECTION.
"attributes not shown
  METHODS method1
    IMPORTING var1 TYPE i OPTIONAL
    value(var2) TYPE i
    EXPORTING var3 TYPE i
    CHANGING var4 TYPE i.
  METHODS method2
    IMPORTING var1 TYPE I DEFAULT 4
    RETURNING value(retval) TYPE i.
PRIVATE SECTION.
"method and/or attribute definition
ENDCLASS.

Method Implementation

Code for functionality of a method placed in **class implementation**.
Parameter names and types are not listed, as they have already been declared in the function definition.

CLASS exampleclass IMPLEMENTATION.
  METHOD method1.
    "code for method1 goes here. In this code
    "var1 and var2 can be used, var3 should be set
    "to a value for export, and var4 will
    "potentially be changed
  ENDMETHOD.
  METHOD method2.
    "similar to above, only var1 can be used and
    RETURN retval. "retval must be returned.
  ENDMETHOD.
ENDCLASS.
**Constructor**

A class may define a constructor method named `constructor`. The method must be `public` and only have `importing parameters`. The constructor is not called explicitly, but rather is called implicitly when a new object is created.

Constructors are useful for:

- Initializing attributes for an object.
- Allocating resources for the object.
- Modifying static attributes (programmatically).
- Triggering other activities that need to happen when a new object is created.

A static constructor method can be created named `class_constructor`. This method will be called first time an object of this class is created or a static attribute or static method of the class is used.

**Self-reference**

Within a method, reference to object attributes or other methods is generally unambiguous. In the event of ambiguity (a parameter has the same name as an attribute), you can refer to the object, its attributes, and its methods using `me`.

Syntax: `me->attributename`
Quick Practice

Object Creation

Objects are created by defining an object reference variable and then creating the object.

The **START-OF-SELECTION** keyword indicates where the runtime environment should begin control flow.

* Object definitions here
START-OF-SELECTION.
DATA objectname TYPE REF TO classname.
CREATE OBJECT objectname.

Further instantiation details (showing parameter) passing shown in the lesson immediately following.