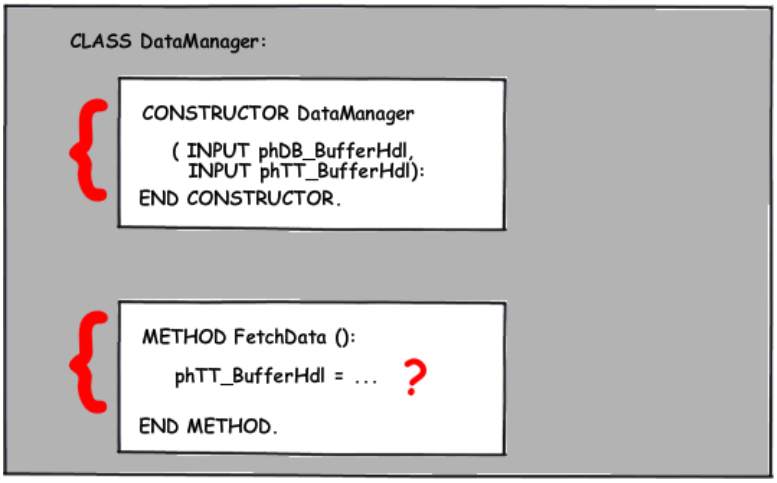
```
DEFINE PRIVATE VARIABLE hDB_BufferHdl AS HANDLE NO-UNDO.
DEFINE PRIVATE VARIABLE hTT_BufferHdl AS HANDLE NO-UNDO.

        CONSTRUCTOR PROTECTED DataManager
            ( INPUT phDB_BufferHdl AS HANDLE,
              INPUT phTT_BufferHdl AS HANDLE ):
            SUPER ().
            ASSIGN hDB_BufferHdl = phDB_BufferHdl
                   hTT_BufferHdl = phTT_BufferHdl.
        END CONSTRUCTOR.
```

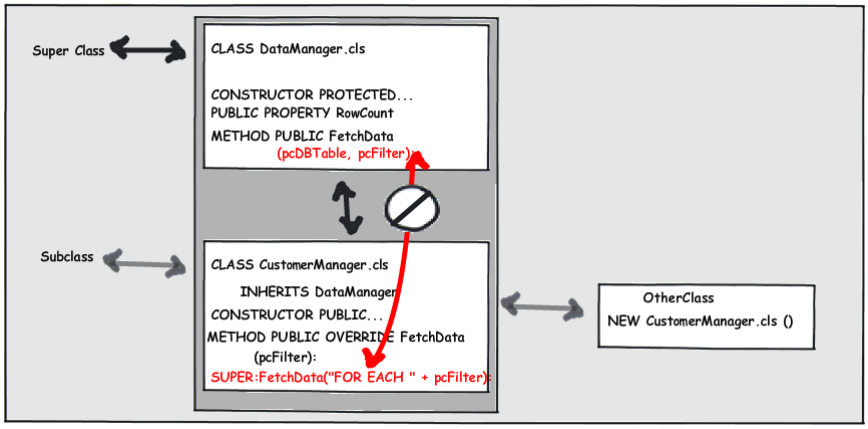
The parameters to the constructor must be assigned to data members in the main block, because just like any other parameters or variables local to a method, the constructor parameters cannot be accessed outside the constructor:



In addition, the **FetchData** method in **DataManager.cls** is changed to turn it into an overloaded method rather than an overridden one. This allows its signature to change so that it accepts two parameters rather than one, which are combined to form the query prepare string. Its return type is changed to **VOID**, since the subclass that invokes it does not need to get back the handle that it previously returned. And because the remaining **FetchData** method in the subclass is the one that must adhere to the **PUBLIC** signature declared in the interface **IDataManager**, the modified version in the super class is made **PROTECTED**:

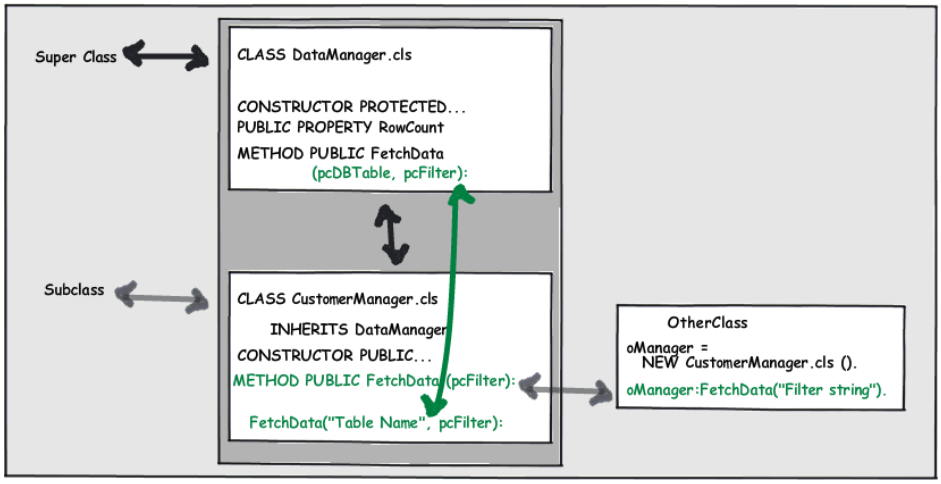
```
    METHOD PROTECTED VOID FetchData
        (INPUT pcDBTable AS CHARACTER ,
         INPUT pcFilter AS CHARACTER ):

        DEFINE VARIABLE hQuery AS HANDLE NO-UNDO.
        CREATE QUERY hQuery.
        hQuery:SET-BUFFERS (hDB_BufferHdl).
        hQuery:QUERY-PREPARE( "FOR EACH " + pcDBTable + " WHERE " + pcFilter).
        hQuery:QUERY-OPEN ().
        hQuery:GET-FIRST ().
        DO WHILE NOT hQuery:QUERY-OFF-END:
            hTT_BufferHdl:BUFFER-CREATE ().
            hTT_BufferHdl:BUFFER-COPY (hDB_BufferHdl).
            hQuery:GET-NEXT ().
        END.
        RowCount = hQuery:NUM-RESULTS.
        hQuery:QUERY-CLOSE ().
        DELETE OBJECT hQuery.
    END METHOD.
```

The subclasses **CustomerManager.cls** (shown here) and **EmployeeManager.cls** are modifed to remove the **OVERRIDE** keyword and the **SUPER** reference when invoking the super class **FetchData**, since it is now an overloaded rather and an overridden method. It would be an error to define a subclass method as an **OVERRIDE** when its signature does not match the signature of the method being overridden.



The compiler distinguishes the two variants now by their differing signatures. The subclass now passes two arguments to the super class version of **FetchData** instead of one:

```
METHOD PUBLIC HANDLE FetchData( INPUT pcFilter AS CHARACTER ):

    FetchData("AutoEdge.Customer", pcFilter).
    RETURN TEMP-TABLE ttCustomer:HANDLE.

END METHOD.
```



In the final presentation on the ***Object Life Cycle***, **CustomerClient.cls** (shown here) and **EmployeeClient.cls** are modified to make their fetch methods **PUBLIC**, and to remove their invocation from the constructor. In this way **FetchCustomers** or **FetchEmployees** can be invoked independently of creating the client object itself.

Also, the filtering value is now passed in as a parameter, rather than being hardcoded, to allow different instances of the objects to retrieve different sets of data.

```
    CONSTRUCTOR PUBLIC CustomerClient (  ):
        SUPER ().
    END CONSTRUCTOR.


    METHOD PUBLIC VOID FetchCustomers(INPUT pcFilterValue AS CHARACTER ).

        DEFINE VARIABLE hBuffer  AS HANDLE NO-UNDO.
        DEFINE VARIABLE oManager AS OOSamples.IDataManager.
        DEFINE VARIABLE cFilter  AS CHARACTER NO-UNDO.
        DEFINE VARIABLE hTable   AS HANDLE    NO-UNDO.

        oManager = NEW OOSamples.CustomerManager().
        cFilter = "CustomerBirthCountry = '" + pcFilterValue + "'".
        hTable = oManager:FetchData(INPUT cFilter).
        hBuffer = hTable:DEFAULT-BUFFER-HANDLE.
        hBuffer:FIND-FIRST ().
        MESSAGE " Customer "
           hBuffer:BUFFER-FIELD ("CustomerFirstName"):BUFFER-VALUE
           hBuffer:BUFFER-FIELD ("CustomerLastName"):BUFFER-VALUE
           " is the first of " oManager:RowCount " Customers with "
           cFilter VIEW-AS ALERT-BOX.
        RETURN.

    END METHOD.
```

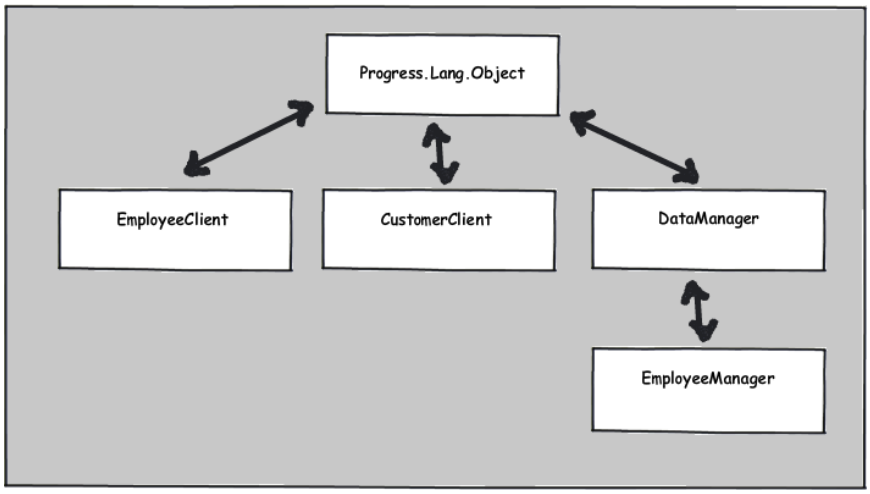
A new class **AutoEdgeManager** now creates instances of both **CustomerClient.cls** and **EmployeeClient.cls**.

```
/*------------------------------------------------------------------------
    File        : AutoEdgeManager

  ----------------------------------------------------------------------*/

USING Progress.Lang.*.

CLASS OOSamples.AutoEdgeManager:
```

The new class defines a temp-table to hold references to objects -- running instances of the other classes that it starts. **ObjectNum** is just a sequence number. **ObjectType** is the name of the client class invoked. **ObjectFilter** is the filter string passed in to qualify the **WHERE** clause applied to the data retrieved into each object. And finally, the **ObjectRef** field holds the object reference to each created (NEW'd) **EmployeeClient** or **CustomerClient** object. As a temp-table field, **ObjectRef** is required to be defined as the type **Progress.Lang.Object**, the virtual parent class for all ABL classes.

```
        DEFINE TEMP-TABLE ttObject
          FIELD ObjectNum AS INTEGER
          FIELD ObjectType AS CHARACTER
          FIELD ObjectFilter AS CHARACTER
          FIELD ObjectRef AS Progress.Lang.Object.
```

All classes are subclasses of **Progress.Lang.Object**, just as **EmployeeManager** is a subclass of **DataManager**:



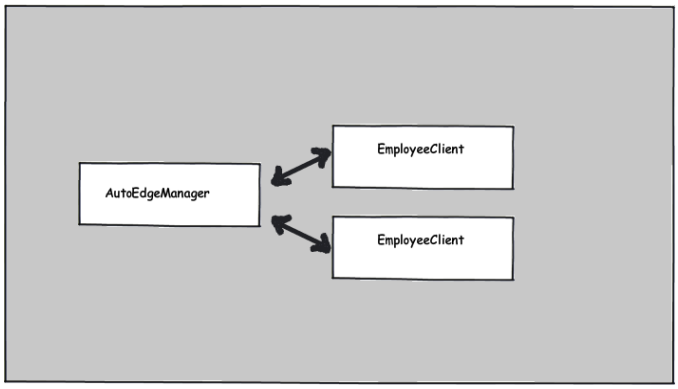
The constructor invokes the class's one method, **StartObjects**.

```
CONSTRUCTOR PUBLIC AutoEdgeManager (  ):
        SUPER ().
        StartObjects().
END CONSTRUCTOR.
```

StartObjects itself creates two instances of **EmployeeClient** and two of **CustomerClient**, which in turn create instances of their respective manager classes to retrieve data. The identifying information for each object, including its object reference, is added to the **ttObject** temp-table.

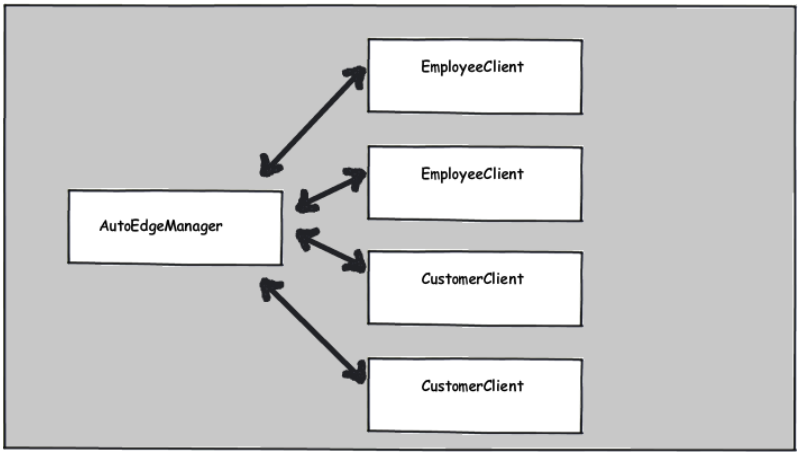
```
METHOD PRIVATE VOID StartObjects(  ):
    CREATE ttObject.
    ASSIGN ttObject.ObjectNum = 1
           ttObject.ObjectType = "EmployeeClient"
           ttObject.ObjectFilter = "SalesRep"
           ttObject.ObjectRef = NEW OOSamples.EmployeeClient().
    CREATE ttObject.
    ASSIGN ttObject.ObjectNum = 2
           ttObject.ObjectType = "EmployeeClient"
           ttObject.ObjectFilter = "Admin"
           ttObject.ObjectRef = NEW OOSamples.EmployeeClient().
```

After these first two groups of statements have been executed, these objects are now in the session:



After the next two sets of statements have been executed, these are now two CustomerClient objects as well as the two EmployeeClient objects:

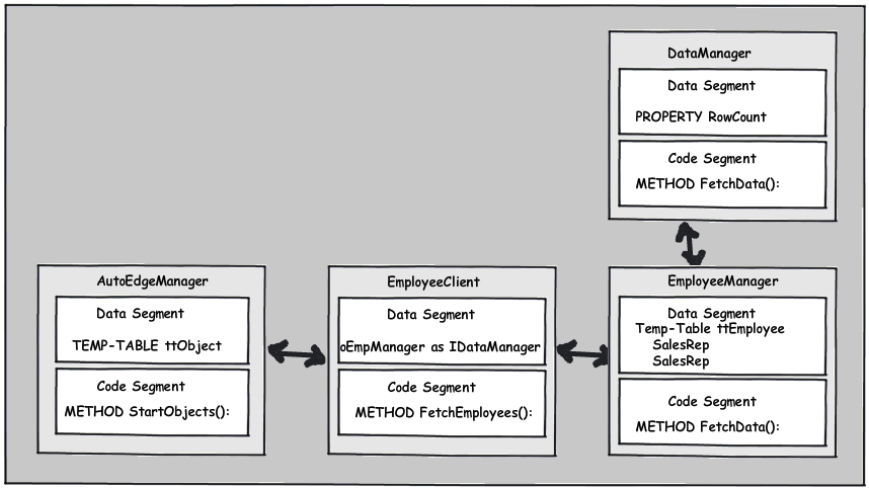
```
CREATE ttObject.
ASSIGN ttObject.ObjectNum = 3
       ttObject.ObjectType = "CustomerClient"
       ttObject.ObjectFilter = "USA"
       ttObject.ObjectRef = NEW OOSamples.CustomerClient().
CREATE ttObject.
ASSIGN ttObject.ObjectNum = 4
       ttObject.ObjectType = "CustomerClient"
       ttObject.ObjectFilter = "Germany"
       ttObject.ObjectRef = NEW OOSamples.CustomerClient().
```



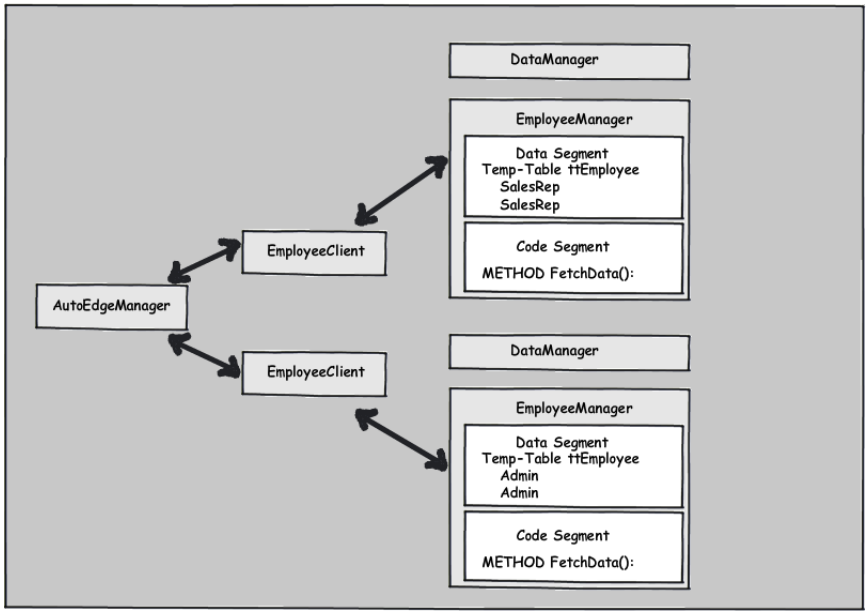
Next, for each row in the temp-table, the method invokes the correct fetch method. Because the **ObjectRef** field is of the generic type **Progress.Lang.Object**, the code must **CAST** the reference to the appropriate type to tell the compiler what type is actually being referenced. This tells the compiler to allow the type-specific methods **FetchEmployees** and **FetchCustomers** to be invoked.

```
FOR EACH ttObject:
    IF ttObject.ObjectType = "EmployeeClient" THEN
        CAST (ttObject.ObjectRef,
           OOSamples.EmployeeClient):FetchEmployees(ttObject.ObjectFilter).
    ELSE IF ttObject.ObjectType = "CustomerClient" THEN
        CAST (ttObject.ObjectRef,
           OOSamples.CustomerClient):FetchCustomers(ttObject.ObjectFilter).
END.
```

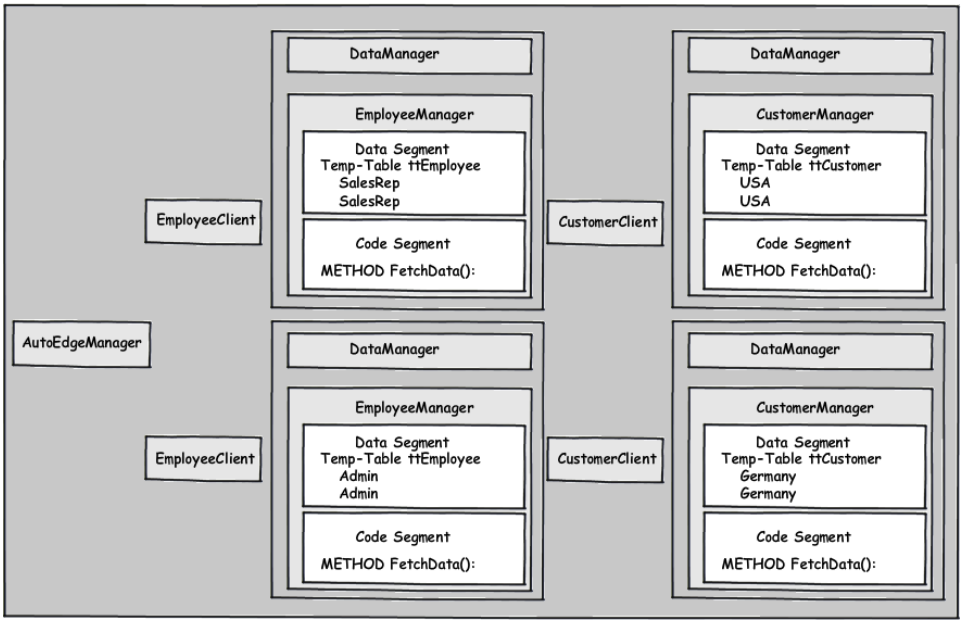
After the first iteration of the **FOR EACH** loop, the **EmployeeClient** object has created a new **EmployeeManager**. The AVM transparently creates an instance of **DataManager** as well, since that is **EmployeeManager's** super class and forms parts of its behavior and data.



After the second iteration of the loop, there are two instances of **EmployeeManager**, each with its own instance of **DataManager** to support it. Runtime optimization may eliminate the duplication of the code segments of these objects, but each has its own data, for instance the different contents of the two **ttEmployee** tables, resulting from two different filter values being passed in to them.



After the final two iterations of the loop, two instances of **CustomerManager** have been created as well, each again with its own **DataManager** object:



The method now deletes each client object it created, which will in turn delete the corresponding manager object. It also deletes each row from the temp-table.

```
        FOR EACH ttObject:
            DELETE OBJECT ttObject.ObjectRef.
            DELETE ttObject.
        END.
        RETURN.

    END METHOD.
```

After the first iteration of the loop, for instance, an **EmployeeClient** object, along with its **EmployeeManager** object and **DataManager** object, will be gone.