



# Object-Oriented Programming Using SAS/AF Software:

An Email Class Example.

by  
Chris Zogby  
Zogby Enterprises



## Outline

- Structured and Object-Oriented Programming Elements
- Benefits of OOP
- Email Class Definition
- Sample Application Code Using Email Class
- Conclusions

## Structured Programming Basics

- Inter-branching Code Blocks
- Branching Statement like GOTO & LINK
- Block Built with Fundamental Coding Units
  - ◆ IF-THEN-ELSE;
  - ◆ DO WHILE(*condition*); END;

## Structured Programming Example

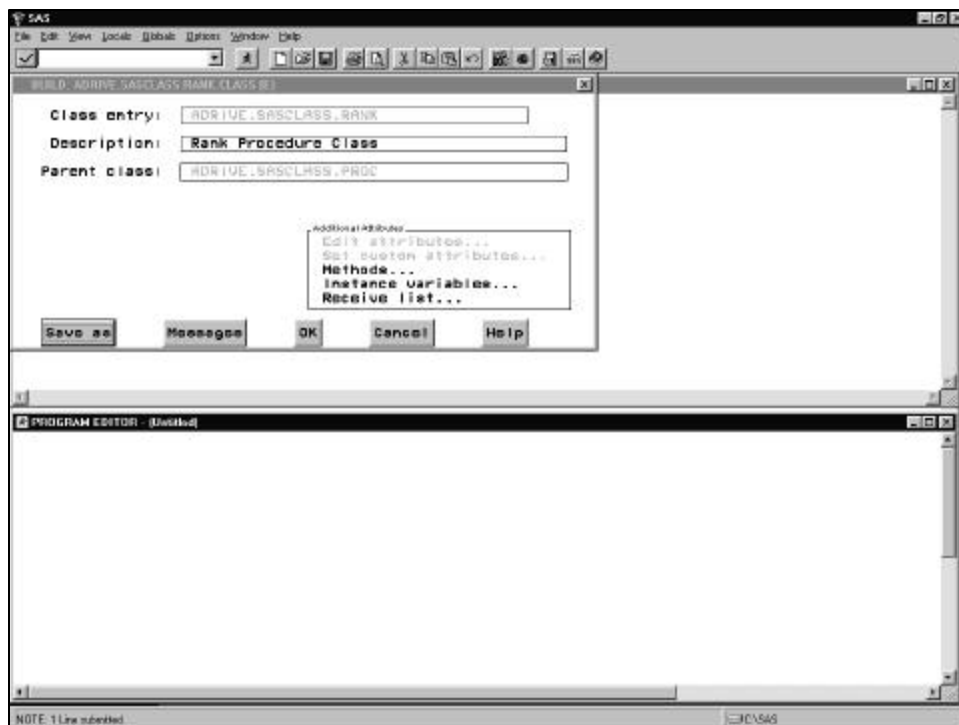
Example written SAS 6.12 screen control language.

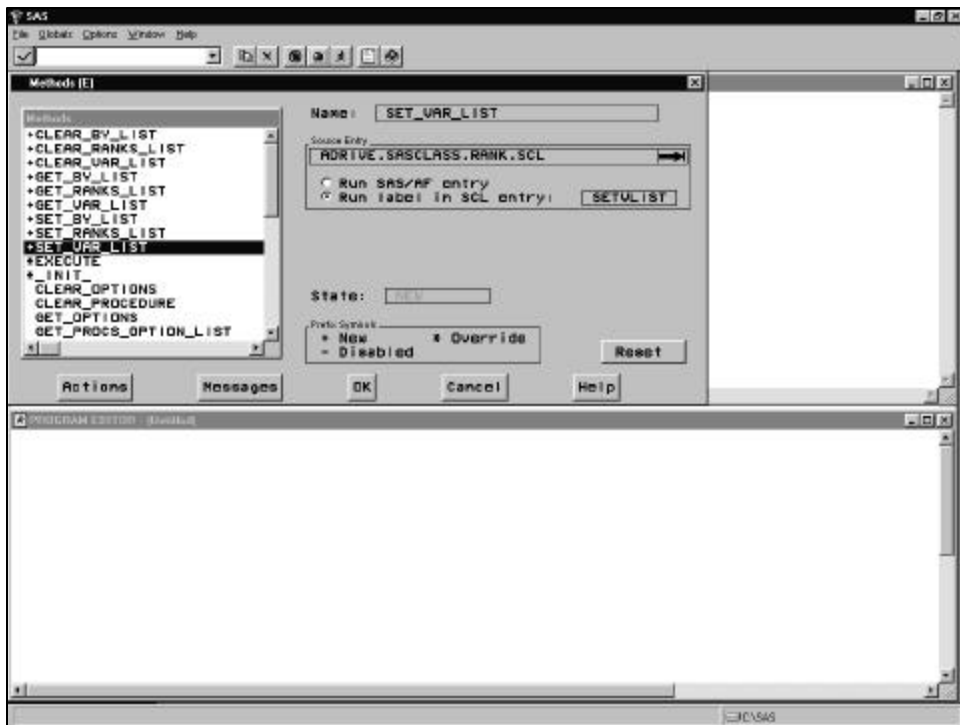
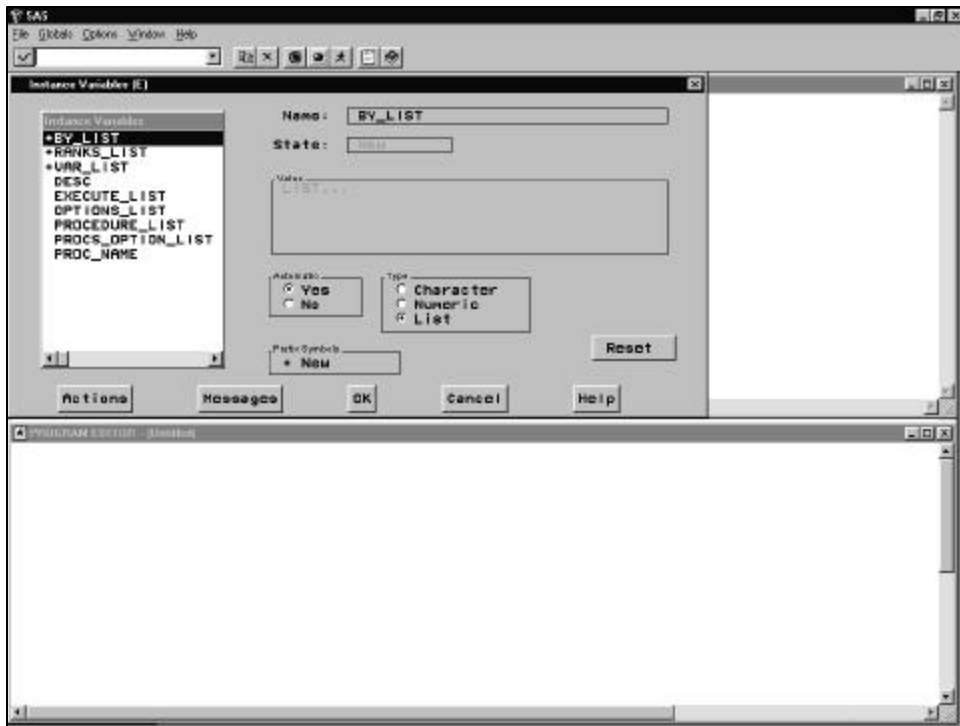
```
INIT:  
IF (  
  Exist(' WORK.NAMES' ,  
        ' DATA' )  
) THEN  
  LINK DEL_DATA;  
Return;  
  
DEL_DATA:  
  RC = DELETE(' WORK.NAMES' ,  
              ' DATA' )  
        );  
Return;
```

# OOP Basics

- Class is Fundamental Programming Unit
- Class Components
  - ◆ Data (\* instance variables)
  - ◆ Operations (\* methods)
  - ◆ Inheritance Relationship
- Object is Fundamental Software Unit
  - ◆ Object is an Instance of its Class

\* SAS/AF version 6.12 terminology





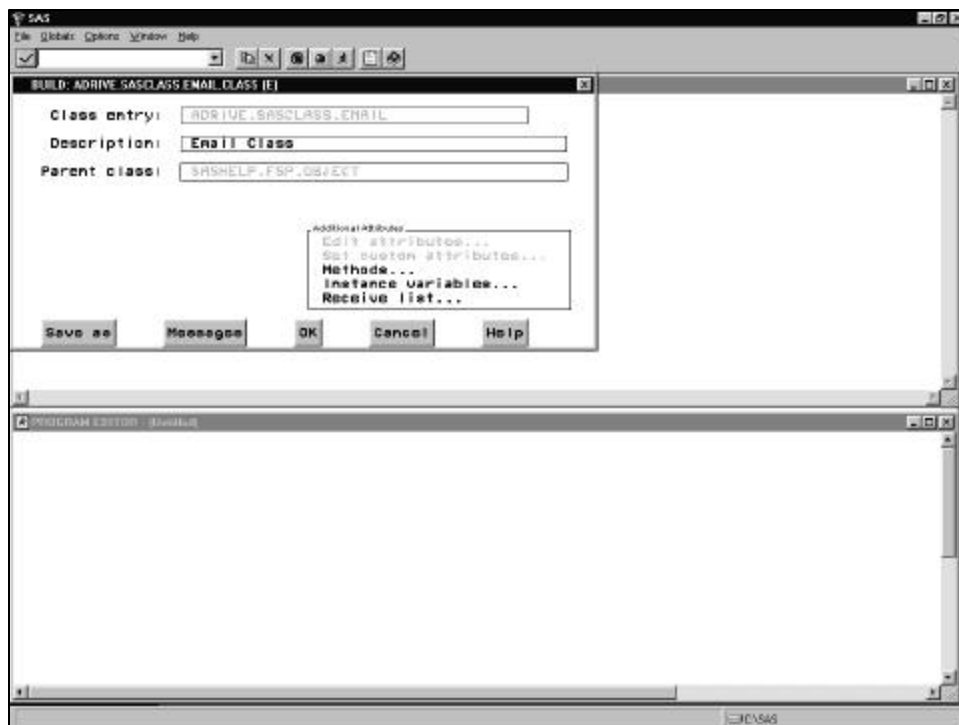
```
00367
00368 *
00369 // SET_VAR_LIST Method Sets Value Of VAR_LIST Instance Variable (LIST).
00370 *)
00371
00372 SETVARLIST:
00373
00374     METHOD LIST 8;
00375
00376     RC = SETNITEHL(PROCEDURE_LIST,COPYLIST(LIST),'VAR STATEMENT');
00377
00378     ENDMETHOD;
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
```

## Benefits of OOP

- Inheritance of Parent Class's Properties
- Encapsulation of Data and Actions
  - ◆ Object's Data and Actions Bound To Object
  - ◆ Data Stored Persistently Until Cleared, Modified, or Object is Destroyed
- Increased Efficiency and Organization
  - ◆ Reduced Code Redundancy
  - ◆ Reduced Application Maintenance Costs

# Email Class Definition

- Parent Class
- Data and Actions Necessary to Send Email
  - ◆ Instance Variables
  - ◆ Methods






## Email Class's Instance Variables

---

---



## Email Class's Methods

---


---

Portion of Email Class's \*\_INIT\_ Method:  
\* Overriden

```
* Call Parents _INIT_;  
CALL SUPER(_SELF_, '_INIT_');  
  
* Populate Master List With Sublists And  
Variables. ;  
MASTER_LIST = INSERTC(MASTER_LIST,  
ATTACHMENT,  
-1,  
'Attachment'  
);  
  
MASTER_LIST = INSERTC(MASTER_LIST,  
SUBJECT,  
-1,  
'Subject'  
);  
  
MASTER_LIST = INSERTL(MASTER_LIST,  
TO,  
-1,  
'To'  
);  
  
MASTER_LIST = INSERTL(MASTER_LIST,  
CC,  
-1,  
'Cc'  
);  
  
MASTER_LIST = INSERTL(MASTER_LIST,  
BODY,  
-1,  
'Body'  
);
```

Portion of Email Class's SEND\_MAIL Method: \* open email file in write mode;

```
fid = fopen('_01email', 'o');  
  
to = getniteml(master_list, 'to');  
cc = getniteml(master_list, 'cc');  
body = getniteml(master_list, 'body');  
  
* write ^to^ recipients;  
if listlen(to) then do;  
rc = fput(fid,  
'!EM_TO! ('  
);  
  
do i = 1 to listlen(to);  
rc = fput(fid,  
getitemc(to, i)  
);  
  
if i ^= listlen(to) then do;  
rc = fput(fid,  
,,  
);  
  
end;  
  
end;  
  
rc = fput(fid,  
,)  
);  
rc = fwrite(fid);  
  
end;
```



## Sample Application Code Using Email Class and Object



### Load Email Class / Create Object

```
* Load Email Class;  
EMAIL_CLASS =  
  LOADCLASS(' LIBNAME. CATALOG. EMAIL. CLASS' );  
  
* Create Instance of Email Class (i. e. Email  
  Object);  
EMAIL_OBJECT = INSTANCE(EMAIL_CLASS);
```

## Create & Set "TO" Recipient List

```
* Create List For TO Recipients;  
TO_ADDRESSES = MAKELIST();
```

```
* Populate List With Email Addresses;  
RC = INSERTC(TO_ADDRESS, 'DICK@AOL.COM', -1);  
RC = INSERTC(TO_ADDRESS, 'JANE@AOL.COM', -1);
```

```
* Set The Object's TO List With TO_ADDRESS List;  
CALL SEND(EMAIL_OBJECT, 'SET_TO', TO_ADDRESS);
```

## Set Attachment File Path

```
* Set The Email Subject Line;  
CALL SEND(EMAIL_OBJECT,  
          'SET_SUBJECT',  
          're: Requested Config File'  
          );
```

```
* Set Configuration File As Attachment;  
CALL SEND(EMAIL_OBJECT,  
          'SET_ATTACHMENT',  
          'C:\SAS\CONFIG.SAS'  
          );
```

## Write Email Body & Dispatch

```
* Create List For Email Message Body;
BODY = MAKELIST();

* Populate List With A Message;
RC = INSERTC(BODY, 'Dear Dick and Jane:', -1);
RC = INSERTC(BODY, ' ', -1);
RC = INSERTC(BODY, 'Please find attached SAS config file.', -1);
RC = INSERTC(BODY, ' ', -1);
RC = INSERTC(BODY, 'Regards, ', -1);
RC = INSERTC(BODY, ' ', -1);
RC = INSERTC(BODY, 'Ed', -1);

* Set Email Message Body;
CALL SEND(EMAIL_OBJECT, 'SET_BODY', BODY);

* Send Email;
CALL SEND(EMAIL_OBJECT, 'SEND_MAIL');
```

## Destroy Object and Loaded Class

```
* Destroy Email Object;
CALL SEND(EMAIL_OBJECT, '_TERM_');

* Destroy Email Class;
CALL SEND(EMAIL_CLASS, '_TERM_');
```

## Conclusions

- Structured Programming and OOP not necessarily mutually exclusive.
- Inheritance & Encapsulation Distinguish OOP from Structured Programming
- Efficiency and Organization Gains Via:
  - ◆ Reduced Code Redundancy
  - ◆ Reduced Application Maintenance Costs

Please Go To NESUG 2001