Lecture 17

External Subprograms

Text:
Chapter 22 (5th Edition)
Chapter 23 (4th Edition)

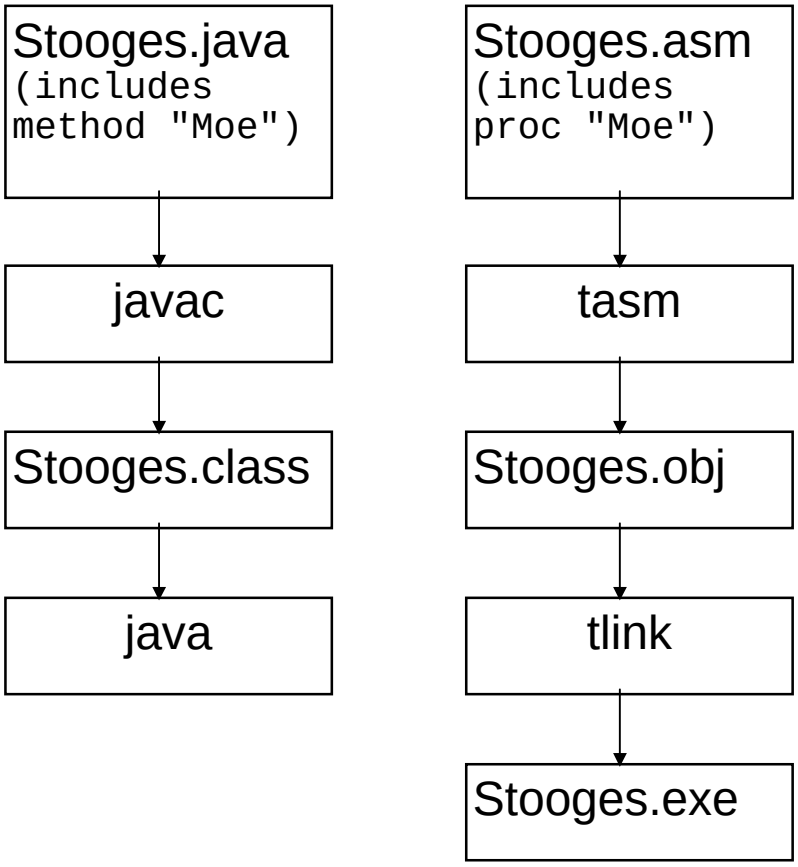
Subprogram

- A smaller part of the "main" program.
- Performs a specific, independent operation
- Data values:
    - global
    - local
    - parameters
- Scoping rules (high-level languages)

```
public class Stooges {

   private int  hammer, nail;
   char actor;
   string name;

   public Stooges (int h,n; char a) {
      hammer = h;
      nails = n;
      actor = a;
   }

   public void Moe ( string says ) {
      SayLine (says);
   }

   private void SayLine (string says) {
      system.out.printline (says);
   }
}
```

A subprogram may be part of the main program text (i.e., in the same file)

It requires one compilation or assembly.

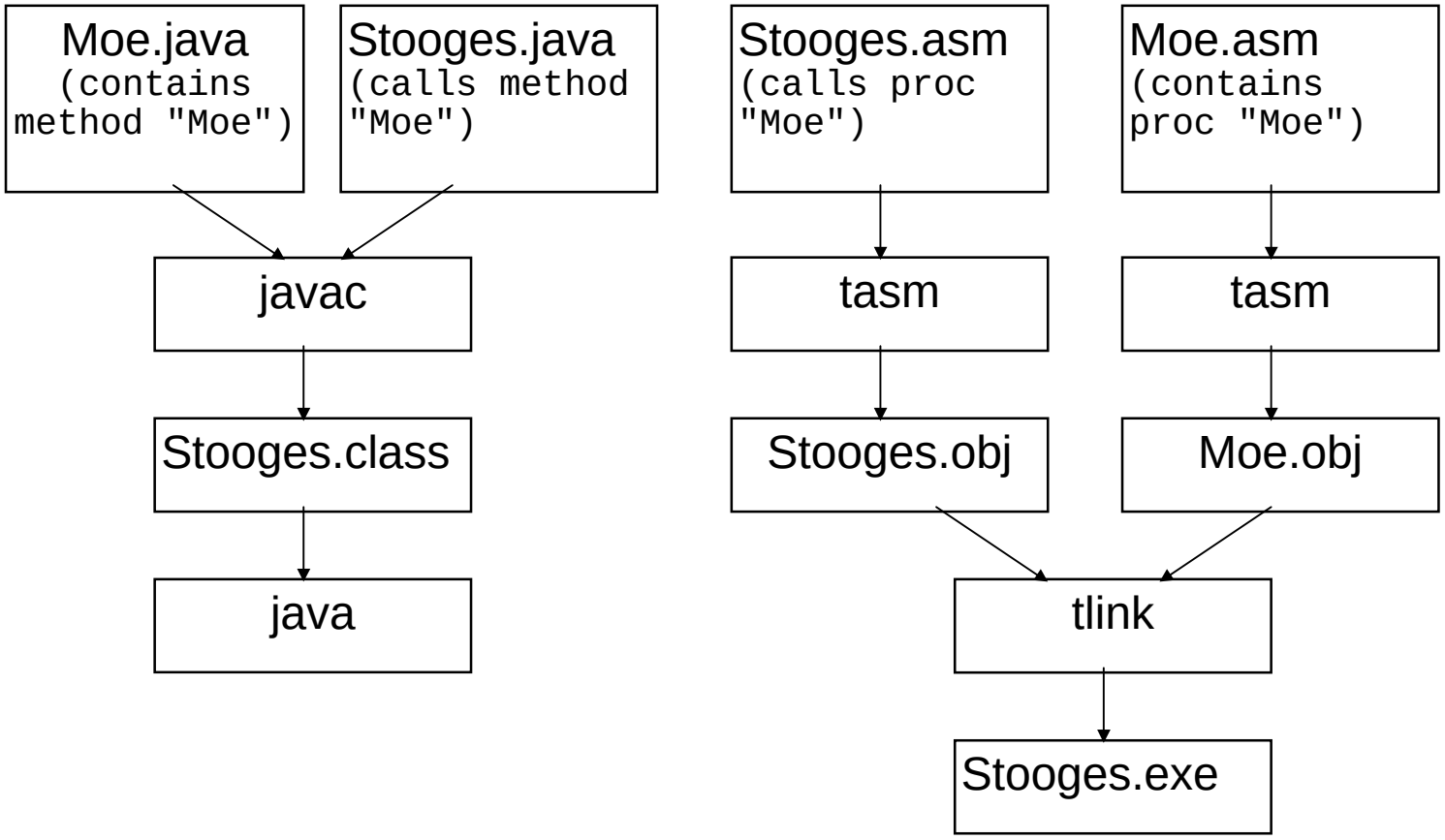


We will refer to this scenario as **INTERNAL Subprograms**

Clearly, data values in the main program are global to the method/proc "Moe".

A subprogram may be separate from the main program text (i.e., in a different file)

It requires two compilations or assemblies.



We will refer to this scenario as

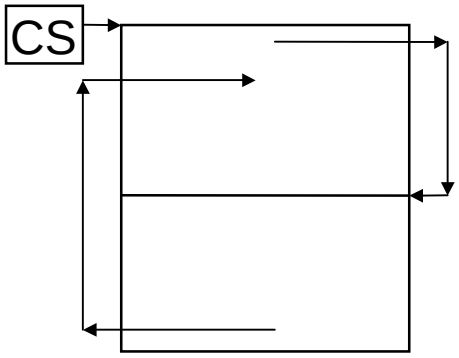
EXTERNAL Subprograms

Clearly, data values in the main program are local to their own methods or procs.

NEAR and FAR calls

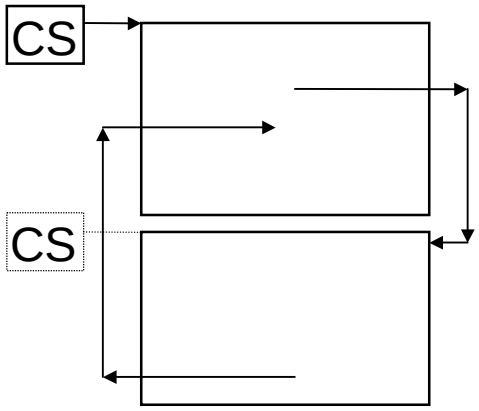
NEAR- call within the same segment

```
            CALL    NEARPROC
              …
NEARPROC    PROC    NEAR
              …
            RET

NEARPROC    ENDP
```



FAR- call to a different segment

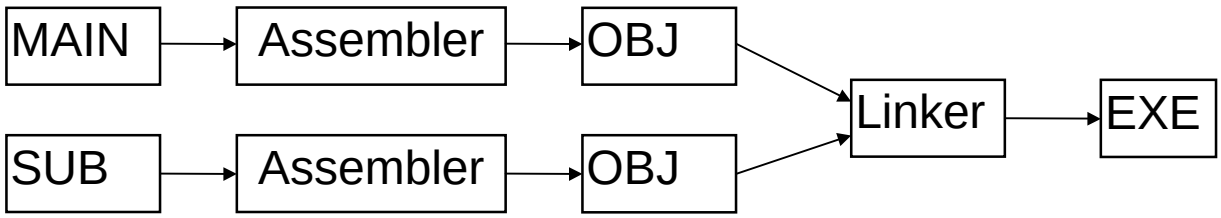
```
            CALL    FARPROC
              …
FARPROC     PROC    FAR
              …
            RET

FARPROC     ENDP
```



A NEAR call requires saving the IP on the stack.

A FAR call requires saving the CS and IP on the stack.

The main program and the external subprogram are put in separate files, and assembled separately.



This introduces several problems.

```
MAIN PROC   FAR
…
     CALL   ADD
…
MAIN ENDP
```

The symbol ADD is not defined in this program. The assembler will generate an error message.

```
ADD  PROC   FAR
…
     RET
…
ADD  ENDP
```

The symbol ADD is local to this program. The linker will not find it.

This is solved with two directives:

EXTRN

Informs the assembler that a symbol will be defined in another program.

PUBLIC

Informs the assembler that a symbol should be available to the public. The symbol is listed in the .OBJ file so that the linker can find it.

```
      EXTRN ADD:FAR
MAIN  PROC  FAR
…
      CALL  ADD
…
MAIN  ENDP
```

```
      PUBLIC ADD
ADD   PROC  FAR
…
      RET
…
ADD   ENDP
```

```
          EXTRN    ADDSUB:FAR
STACKSG   SEGMENT  PARA  STACK 'Stack'
          DW       32
STACKSG   ENDS
DATASG    SEGMENT  PARA 'Data'
WORDS     DW       1,2,3,4,5
DATASG    ENDS
CODESG    SEGMENT  PARA 'Code'
BEGIN     PROC     FAR
          ASSUME   SS:STACKSG,DS:DATASG,CS:CODESG
          MOV      AX,DATASG
          MOV      DS,AX
;
          CALL     ADDSUB
;
          MOV      AX,4C00H
          INT      21H
BEGIN     ENDP
CODESG    ENDS
          END      BEGIN
```

```
          PUBLIC   ADDSUB
STACKSG   SEGMENT  PARA  STACK 'Stack'
          DW       32
STACKSG   ENDS
DATASG    SEGMENT  PARA 'Data'
DATASG    ENDS
CODESG    SEGMENT  PARA 'Code'
ADDSUB    PROC     FAR
          ASSUME   SS:STACKSG,DS:DATASG,CS:CODESG

          RET      ; just return
ADDSUB    ENDP
CODESG    ENDS     ; note there is no label here
          END
```

To run these two programs,

Assemble the main:

```
C:>TASM  MAIN
```

Assembles MAIN.ASM
Produces MAIN.OBJ

Assemble the subprogram:

```
C:>TASM SUB
```

Assembles SUB.ASM
Produces SUB.OBJ

Link the two together:

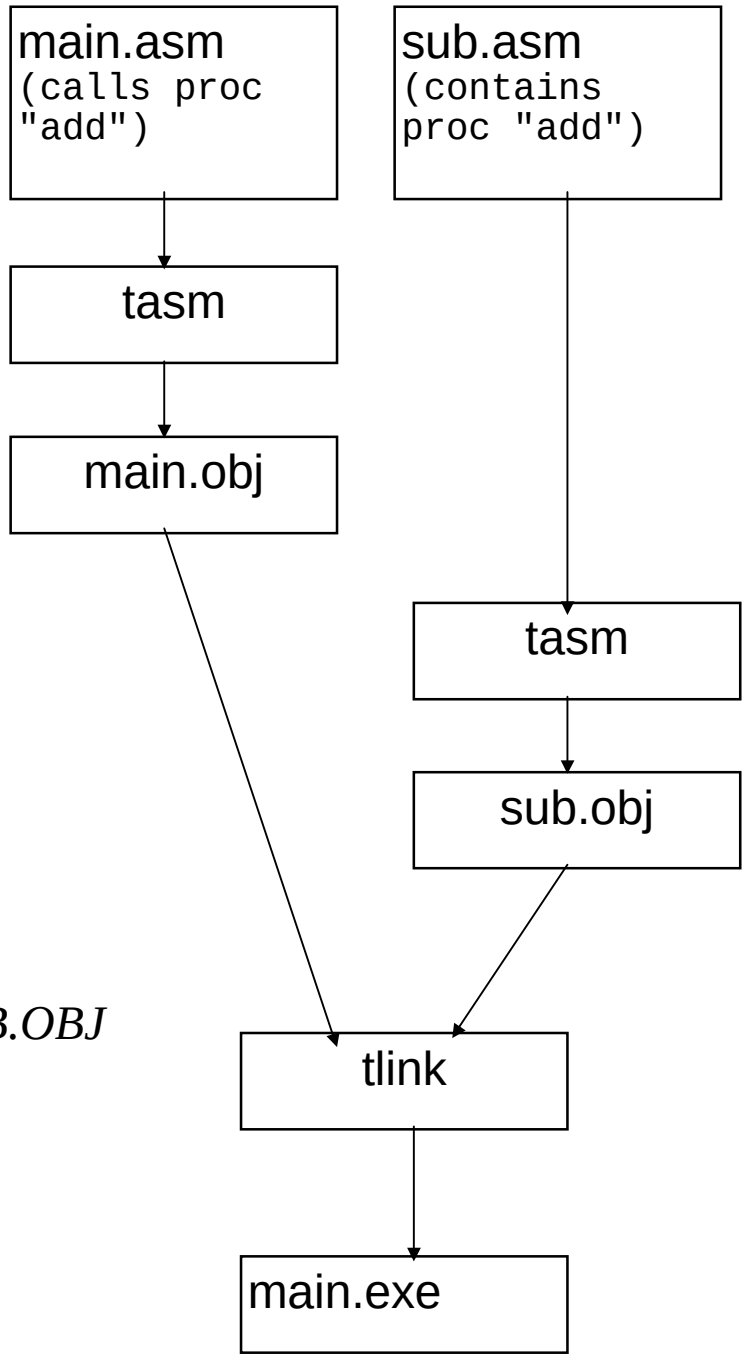
```
C:>TLINK MAIN+SUB
```

Links MAIN.OBJ and SUB.OBJ
Produces MAIN.EXE

Run the program

```
C:>MAIN
```

Executes MAIN.EXE



Sharing common data between programs

Suppose a main and subprogram want to use the same translation table (rather than define the bytes twice)

Main program:

```
            EXTRN     SUBPROG
            PUBLIC    TRANSTBL
Data section
TRANSTBL    GENDATA   20,40,100

MAIN        PROC      FAR
```

Subprogram:

```
            EXTRN     TRANSTBL:BYTE
            PUBLIC    SUBPROG

SUBPROG     PROC      FAR

            MOV       AL,TRANSTBL[SI]
```

This will cause the assembler to use the offset for **TRANSTBL** as determined in the main program for the MOV instruction in the subprogram. The use of the SI register in the MOV instruction implies the use of the DS register. The DS register still points to the data segment of the main program.

Exercises - Lecture 17

Given the following main program, write an external subprogram that will sum the elements of the array "Numbers" and place that sum in the variable "Sum"
Supply the appropriate EXTRN and PUBLIC directives for both programs.

```
          EXTRN
          PUBLIC
STACKSG   SEGMENT  PARA  STACK 'Stack'
          DW       32
STACKSG   ENDS
DATASG    SEGMENT  PARA 'Data'
Numbers   DW       2,8,4,6,10,12
Sum       DW       ?
DATASG    ENDS
CODESG    SEGMENT  PARA 'Code'
BEGIN     PROC     FAR
          ASSUME   SS:STACKSG,DS:DATASG,CS:CODESG
          MOV      AX,DATASG
          MOV      DS,AX
          CALL     SumArray
          MOV      AX,4C00H
          INT      21H
BEGIN     ENDP
CODESG    ENDS
          END      BEGIN
```

```
          PUBLIC
          EXTRN
STACKSG   SEGMENT  PARA  STACK 'Stack'
          DW       32
STACKSG   ENDS
DATASG    SEGMENT  PARA 'Data'
DATASG    ENDS
CODESG    SEGMENT  PARA 'Code'
ADDSUB    PROC     FAR
          ASSUME   SS:STACKSG,DS:DATASG,CS:CODESG
```

```
          RET
ADDSUB    ENDP
CODESG    ENDS
          END
```