Can a COBOL procedural program be INVOKED with an array of objects instead of a standard CALL with USING?

If you prefer to use OOP syntax to invoke a COBOL procedural program and pass an array of objects as parameters rather than the standard COBOL CALL statement with USING, isCOBOL provides a feasible approach.

First, the isCOBOL class com.iscobol.java.IsCobol enables the INVOKE syntax.
Second, the Java class java.lang.Object[] allows you to pass an array of objects.

An advantage of using an array of objects is that each element can reference a data item with a different type and size.

This program exemplifies it by using two parameters, one alphanumeric and one numeric with computational usage.

```
program-id. MAIN.

configuration section.
repository.
   class isc     as "com.iscobol.java.IsCobol"
   class arrobj  as "java.lang.Object[]"
   .

working-storage section.
01  par1     pic x(5).
01  par2     pic 9(5) comp.
01  rc       pic s99.

01  o-arrobj object reference arrobj.

procedure division.
main.
   move "aa" to par1
   move 1    to par2
   display "STANDARD CALL"
   call "PCALLED" using par1
                        par2
                  giving rc
   display "rc=" rc
```

```
move "bb" to par1
move 2    to par2

set o-arrobj = arrobj:>new(2)
set o-arrobj(0) = par1
set o-arrobj(1) = par2
display " "
display "INVOKE"
invoke isc "call" using "PCALLED"
                        o-arrobj
                giving rc
display "rc=" rc

move "cc" to par1
display " "
display
"INVOKE with direct syntax, using previous array of objects"
move 3    to par2
perform 3 times
  set rc = isc:>call ("PCALLED", o-arrobj)
  display "rc=" rc
  compute par2 = par2 * 2
end-perform

display "END"
goback.
```

Once each element in the array of objects references a data item, you can modify the original data items and then pass the array to the program using any of the INVOKE OOP syntaxes shown in the sample program.

On the other hand, the called COBOL procedural program does not need anything special for this to work. The following is an example of it.

```
program-id. PCALLED.

working-storage section.
01  w1   pic x(5).
01  w2   pic 9(5).
01  rc   pic s99.

linkage section.
01  par1   pic x(5).
01  par2   pic 9(5) comp.
```

```
procedure division using par1 par2.
main.
    move par1 to w1
    move par2 to w2
    display "received: par1=" w1 " par2=" w2
    compute rc = w2 + 1
    goback rc.
```

To test the sample programs, compile them using: iscc main.cbl pcalled.cbl
Then, run the main program with: iscrun MAIN