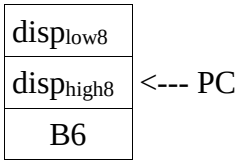
Sequence of steps for INVOKEVIRTUAL

The instruction format and what it means

INVOKEVIRTUAL disp (16 bits)

here **disp** is a 16-bit unsigned integer that indicates the position in the constant pool (accessible by CPP) that contains the starting address within the method area of the method being invoked. When stored as a machine instruction in the method area of memory, disp will be split into two 8-bit bytes and stored as a big-endian value.

Method area when invoked



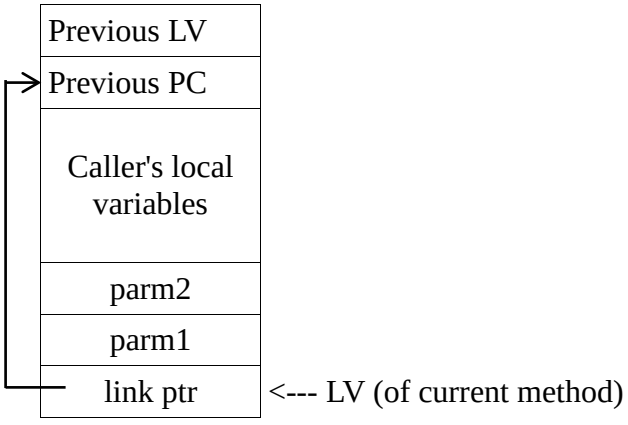
Prior to invoking a method:

1. Caller pushes onto the stack a reference OBJREF to the object being called. This also serves as parameter 0 (or $parm_0$)

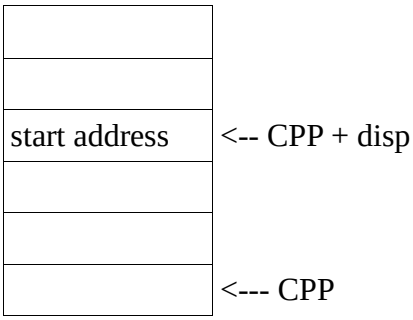
   - See the discussion in your book on page 263 for why this is not really needed for IJVM but is retained for consistency with the Java Virtual Machine (JVM).

2. The caller pushes the remaining parameters onto the stack – say $parm_1$, $parm_2$..$parm_n$.

The local variable area (referenced by LV) before INVOKEVIRTUAL is called. The current method is presumed to have had 2 parameters. INVOKEVIRTUAL will have to construct something similar for the called method.



Constant pool area when INVOKEVIRTUAL is called,



Here: #parms is a 16 bit number giving the number of parameters; the parameters are presumed to have been previously pushed onto the stack. Recall that $parm_0$ represents OBJREF.

Sequence that occurs for INVOKEVIRTUAL:

1. The two ***unsigned*** index bytes that follow the op-code are used to construct an index into the constant pool (with the first value representing the high-order byte of this index).

2. The instruction computes the base address of the new local variable frame by subtracting off the number of parameters from the stack pointer and setting LV to point to OBJREF.

3. At this location the implementation stores the address of the location where the old PC is to be stored, overwriting OBJREF.

    - This address is computed by adding the size of the local variable frame to the address contained in LV

4. Immediately above the address where the old PC is to be stored is the address where the old LV is to be stored.

5. Immediately above that address is the beginning of the stack for the newly called procedure.
    - SP is set to point to the old LV, which is the address immediately below the first empty location on the stack.

    - Our stacks always grow upward, toward higher addresses.

6. The PC is set to point to the 5^{th} byte in the method code space.

See page 264 for a graphic of the changes that occurred.