Use the memory addresses in the picture above to describe the steps during execution of compiled code that occur when the line \texttt{tmpA2.m2();} is invoked in \texttt{foo}, assuming that this \texttt{m2} is compiled. How is this different from if \texttt{A.m1()} was invoked?

- In bytecode, \texttt{tmpA2.m2()} translates to: <push obj reference on operand stack>; invokevirtual \#Y //class A (tmpA2’s type), method m1()V; ; (Y is index into constant pool)

1. Get the value (address) in variable \texttt{tmpA2}, call it ADDR
2. Go to memory at this address with offset 0 (offset to obj’s internal representation and VMT): Mem[0x1060+0] -> 0x777
3. Lookup the offset of \texttt{m2} in Class A (the type of variable \texttt{tmpA2})
   - Note that because we layout the fields and methods from highest in the hierarchy to the lowest – we are guaranteed that \texttt{m2} is at the same offset in a child object (any subtype of \texttt{A}) as it is in an object of type \texttt{A}
   - Call it \texttt{OFFSET}, in the solution above, \texttt{OFFSET} is 4
4. Get address of method from memory at the value from step 2 plus \texttt{OFFSET}: 0x777+4 -> 0x789 if not compiled, 0x1040 if compiled
5. Call method: call 0x1040

- If method is not compiled, then the call is made to address 0x789. This is the compiler or interpreter stub: code that gets invoked to compile the bytecode method (and store it then invoke it) or to execute the bytecode method instruction-by-instruction via the interpreter.
- If \texttt{A1} is invoked (see next page)
Followup Question

• Use the memory addresses in the picture above to describe the steps during execution of compiled code that occur when the line `tmpA2.m2();` is invoked in foo, assuming that this m2 is compiled. How is this different from if A.m1() was invoked?
  – When A1 is invoked, we do so in source code via: A.m1(); which in bytecode is `invokestatic #X //class A, static method m1()V;` (X is index into constant pool)

1. Get the value (address) of the global statics table in the virtual machine process
2. Lookup the offset of A.m1 in statics table map
   – Call it OFFSET, in the solution above, OFFSET is 8
3. Get address of method from memory at the value from step 1 plus OFFSET: 0x444+8 -> 0x789 if it has not been compiled yet (else this value would have been updated with a different value)
4. Call method: call 0x789

• In this case, the call is made to address 0x789 the very first time this instruction is encountered during execution. This is the compiler or interpreter stub: code that gets invoked to compile the bytecode method (and store it then invoke it) or to execute the bytecode method instruction-by-instruction via the interpreter.
  – If the VM is using a compiler, when it “stores it” above, it means that we go to memory location 0x444+8 and replace the value 0x789 with the address of the compiled method for m1 (not shown in the example), call it 0x1200
  – The next time the instruction is encountered during execution, the same steps above are followed, only the call in step 4 is: call 0x1200