Internal Runtime Representations

For the code below, draw (on the next page) the runtime statics table and its map, the internal class representations of each class and their maps, and the object layouts for the two local variables in main.

Assume that we are executing at the end of B.foo() and about to return and that we got here by executing main in class A the first time. That is, figure out which classes are loaded and which methods are compiled (draw bodies and give addresses of), using this execution path. Assume that the header of an object is 2 words wide (word=4bytes).

Use these addresses:
- 0x789 compiler stub method
- 0x999 class A’s internal representation (VMT)
- 0x888 class A’s class map
- 0x777 class B’s internal representation (VMT)
- 0x666 class B’s class map
- 0x555 the VM’s statics table
- 0x444 the VM’s internal map

For each entry, specify the following:
- index, name, value, one of: sm, im, o
  - sm = static method
  - im = instance method
  - o = something other than sm or im

Note that this is Java: among other things all Java fields are given default values by the VM if not initialized.

Assume that the object header size is two words (8bytes) in total. Specify the class name of the member if ambiguous. Use the tables below and add any other memory objects, as needed. Number of entries in tables may not match the solution (there may be more/less).

class A {
    static int field1 = 4;
    int field2 = 3;
    int field3 = 7;
    static int field4;
    static void m1() {...}
    void m2() {...}
    void m3() {...}
    A() {...}
    public static void main(String a[]) { B.foo(); }
}

class B extends A {
    static int fielda = 4;
    int field2;
    int fieldc = 2;
    void m2() {...}
    static void m4() {...}
    void m5() {...}
    B() {...}
    static void foo() {
        A tmpA1 = new A();
        A tmpA2 = new B();
        tmpA2.m2();
    }
}