Understanding the concept of **static**

We explain the concept of **static** and the mechanics of using static variables and methods.

The container to hold objects and static components of a class

At the bottom of the page is a class C. It has declarations of static variable b, static method m, and instance method p. This means that, during execution of a program that uses class C, there will be only ONE copy of b and one copy of m, but p will appear in each object of class C.

Think of having during runtime a container, a box, that contains all the objects of class C as well as the one copy of b and m, as shown to the right below. This container and m and b are created at the beginning of execution, so b and m are present during all of execution. When objects of class C are created, they are placed in the container.

Referencing static components

Other classes can use b and m, since b and m are declared to be public. Since the container is named C, code in the other classes can refer to b and m using:

C.b and C.m(arguments)

The "C." before the name of the field or method call indicates where to look for the field or method —in the container for class C.

Instance method p resides in all objects of class C, as shown. By the inside-out rule, one can call method m from the body of p using simply m(arguments). Similarly, by the inside-out rule, one can refer to static variable b using just the name b.

Comments

The actual implementation at runtime will not have a separate, contiguous part of memory to contain C's objects and static components. But this conceptual view helps us understand the concept and mechanics of **static**.

Also, we have not given examples that show why one might want to *use* static variables and methods.

Here is one idea. The only reason to place a method in each object is so that it can refer to the object's fields and methods.

Here is an example, suppose we want to add a function max(x, y) to class C. Function max will refer only to parameters x and y and not to fields and methods that are in each object of C. Therefore, there is no need to place max in each object —in fact, that would be wasteful. Make max static, so there is only one copy of it.

```
public class C {
   public int b= 5;

   public static void m() {…}

   public void p() { … }
}
```

C: Container for C's objects and static components