The Frame for a call

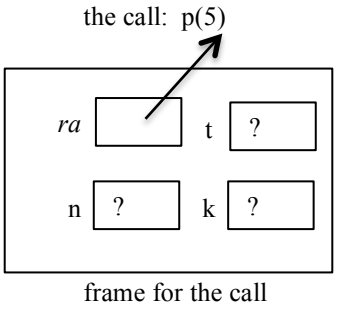
David Gries and Scott Wehrwein

Consider this procedure p, which prints the integers in the range 1..n. We have deliberately put local variable t in the body of a loop, although it is unnecessary, in order to make a point later on.

```
/** Print the integers in 1..n. */
public static void p(int n) {
   for (int k= 1; k <= n; k= k+1) {
      int t= k;
      System.out.println(t);
   }
}
```

Now consider the call p(5). When this call is executed, a box is created, called the *frame for the call*, to contain all information needed to execute the call. This information includes:

1. The parameters of the call (in this case, n),
2. The local variables declared in the method body (k and t), and
3. A *return address* —something to indicate where the call occurred in the program, so it is known where to continue after the call is finished.



frame for the call

We show the return address as a pointer, named *ra*, to just after the call — that's how *we* will show it when *we* are executing a call. The computer may do it differently when the computer is executing a call.

Note that at all three variables initially contain a query, or question mark, because these variables are not initialized to anything in particular when the frame is first created. This is different from a field of a newly created object, which contains a default value that depends on the type of the field.

Other information may be necessary

Other information may be necessary. For example, some methods, like setHour, appear in objects, like object t. The call then names the object that contains the method being called.

In this case, the frame for the call contains not only the parameters, local variables, and return address but also a pointer to the object in which the method resides, so that the fields and methods of the object can be used.

To keep things simple, we will not deal with this case. Our examples will use only methods that contain only parameters and local variables and not fields.

Learning to infer from knowledge of execution

One learns how a statement is executed because it allows you to draw important conclusions from it. We have not completed the algorithm for executing a method call, but only its first step. Yet, think of the following. The creation of the frame for the call creates the parameters and local variables —if the computer is executing the call, it allocates space for the parameters and local variables. We leave you, then, to answer the following question.

1. When during a call of procedure p is space allocated for local variable t?
2. s it allocated every time the repetend is executed and de-allocated when the loop body is finished?