Figure 5.2 A first look at *static*.

The keyword **static** has more than one use in Java. Here's a first look at it.

Sometimes we seem to need to mention an object from a class, and sometimes just the class itself. For example, consider

String s = "Hello World!"; **int i = s.length();** The method **length()** tells us something about the particular object **s**. We need to say **s.length()**, because **String.length()** would be ambiguous.	**versus**	**String s = "123";** **int i = Integer.parseInt(s);** The method **parseInt(String s)** is used for *all* integers, so we affixed the class name to it. (We'll soon see that there are no actual **int** "objects",)

When writing a method, the keyword **static** says that the method belongs to the class as a whole, and not to any particular object.

- **parseInt** is declared **static** because the **Integer** class has no objects, per se.
- **main** is declared **static** because at the outset of program execution, there are no objects to invoke methods upon

When you declare an instance variable **static**, you are saying that the variable belongs to the class as a whole, and not to any particular object.

- this means there is only *one copy* of the variable
- useful for storing information used by various **static** methods

```
public class TestInput {
    private static BufferedReader keyb;
    public static void main (String[] args) {
        keyb = new BufferedReader(new InputStreamReader(System.in));
        String inputLine = keyb.readLine();
        otherReader();
    }
    private static void otherReader () {
        String inputLine = keyb.readLine();
    }
}
```