The verification algorithm will reject this method, because it can't be sure of a constant stack height. If it permitted this method to execute, the stack would grow by 1 each time. Eventually, the program might overflow the stack. Because you don't want that to happen, the verification algorithm rejects this code.

6.5.4 Example 4: Dealing with Subclasses

One more complication: it isn't necessary for two stack pictures to be identical when two different flows of control come to the same place. Here's a (somewhat contrived) example. The example depends on three classes:

```
abstract class Person {
   abstract void printName();
}

class Programmer extends Person {
   void printName() { /* Implementation goes here */ }
}

class Author extends Person {
   void printName() { /* Implementation goes here */ }
}
```

The code we wish to verify is

```
.method public static print(ZLProgrammer;LAuthor;)V
   iload_0                    ; Is the boolean false?
   ifeq false                 ; If not,
   true: aload_1              ; then push the programmer
   goto print
   false: aload_2             ; Otherwise, push the author
   print: invokevirtual Person/printName ()V
                              ; Call printName on the Person
                              ; This works whether it's an
                              ; Author or a Programmer,
                              ; since each is a Person
   done: return
.end method
```

This method takes three arguments: a `boolean` control, a `Programmer`, and an `Author`. If the control is `true` then it prints the name of the `Programmer`. Otherwise, it prints the name of the `Author`. The program arrives at `print` with either