

07.03 Virtual Lecture Notes

Java permits instance variables and local variables to have the same name, as indicated in the following two examples. In both cases, it appears that the intent is to assign the value passed to a parameter called `name` to a private instance variable also called `name`.

```
private String name;           private String name;

public Student(String name)    public void setName(String name)
{                               {
    Name = name;               Name = name;
}
```

This seems like it would work, so let's try..

- Open the **ThisDemoATester.java** class.
- Examine the source code and notice the instance variable and the local parameter variables have the same name.
- Can you predict what will be printed?
- Run the program and observe the output.

Did you get the results you expected? Java compiles and runs the code without complaint. But the lack of syntax or runtime errors does not guarantee correct output. Instead of displaying the name of a famous cryptographer, `null` is printed, indicating the `name` variable is defined, but not yet assigned a value.

Let's try to disambiguate the situation by returning to familiar territory.

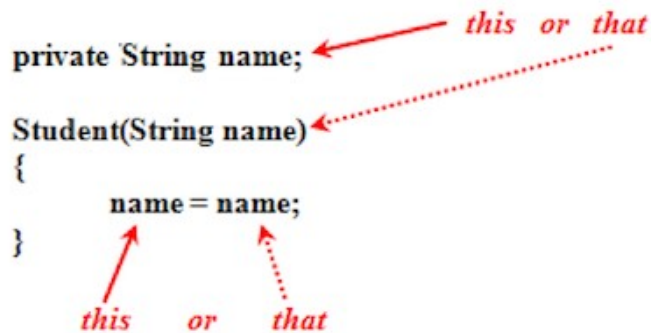
- Open the **ThisDemoBTester.java** class.
- Examine the source code and notice the use of a prefixed identifier for the instance variable `myName`.
- Can you predict what will be printed?
- Run the program and observe the results.

Did you correctly predict the results? Once again, Java compiles with no syntax or runtime errors *and*

the output is what you would expect.

Through some simple experimentation, you have discovered that Java permits instance variables and local variables to have the same name, but the results are not what you might expect.

When the code of **ThisDemoATester** is compiled, Java evaluates which `name` variable should be used for which purpose. In effect, it is trying to determine whether to use *this* one or *that* one, as illustrated below.



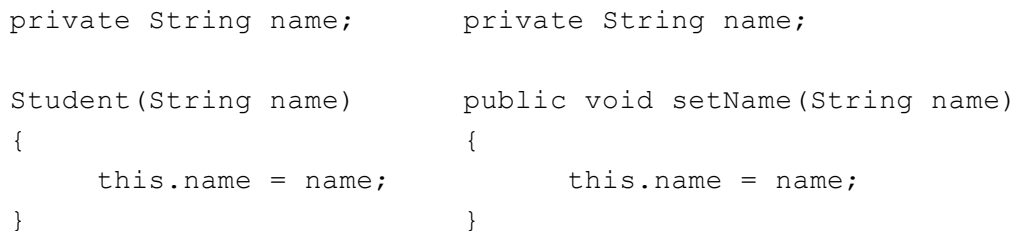
```
private String name;
Student(String name)
{
    name = name;
}
```

The compiler resolves this ambiguous situation by using the most locally-defined variable, in this case the constructor's parameter.

Consequently, the statement `name = name` simply results in the object stored in the local parameter variable (i.e., *that one*) being reassigned to itself!

Therefore, the instance variable (i.e., *this one*) never gets assigned anything, so its initial value remains `null`.

Before you decide that prefixed variable names are the only way out of this dilemma, let's make a minor modification by utilizing the keyword `this`, as shown below:



<code>private String name;</code>	<code>private String name;</code>
<code>Student(String name)</code>	<code>public void setName(String name)</code>
<code>{</code>	<code>{</code>
<code>this.name = name;</code>	<code>this.name = name;</code>
<code>}</code>	<code>}</code>

This minor change resolves the confusion between whether we are referring to the private instance variable (i.e., *this one*) or the local parameter variable (i.e., *that one*).

- Open **ThisDemoCTester.java** class.
- Examine the source code and notice the use of the keyword `this`.
- Can you predict what will be printed?
- Run the program and observe the results.

Notice that `this` is used to assign the local variable value to the private instance variable, but not in its declaration.

The use of the keyword `this` is completely optional unless you use the same identifier name for instance and local variables. It serves the same purpose as prefixed variable names, so you are free to use either approach. Although `this` is implied and does not need to be typed, if you use `this` to identify an instance variable in the constructor, you should use it explicitly throughout the class whenever this instance variable is used in a method to increase clarity. For example,

```
public String getName( )  
{  
    return this.name;  
}
```

Specifically using `this` forces you to recognize that `name` is an instance variable and not a local variable.

