Variable and Method Hiding in Java

**1. Introduction**

In this tutorial, **we’re going to learn about variable and method hiding in the Java language**.

First, we’ll understand the concept and purpose of each of these scenarios. After that, we’ll dive into the use cases and examine different examples.

**2. Variable Hiding**

**Variable hiding happens when we declare a property in a local scope that has the same name as the one we already have in the outer scope.**

Before jumping to the examples, let’s briefly recap the possible variable scopes in Java. We can define them with the following categories:

* local variables – declared in a piece of code such as methods, constructors, in any block of code with curly braces
* instance variables – defined inside of a class and belong to the instance of the object
* class or *static* variables – are declared in the class with the *static* keyword. They have a class level scope.

Now, let’s describe the hiding with examples, for each individual category of variables.

**2.1. The Power of Local**

Let’s have a look at the *HideVariable* class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public class HideVariable {        private String message = "this is instance variable";        HideVariable() {          String message = "constructor local variable";          System.out.println(message);      }        public void printLocalVariable() {          String message = "method local variable";          System.out.println(message);      }        public void printInstanceVariable() {          String message = "method local variable";          System.out.println(this.message);      }  } |

Here we have the *message*variable declared in 4 different places. The local variables declared inside of the constructor and the two methods are shadowing the instance variable.

Let’s test the initialization of an object and calling the methods:

|  |  |
| --- | --- |
| 1  2  3  4 | HideVariable variable = new HideVariable();  variable.printLocalVariable();    variable.printInstanceVariable(); |

The output of the code above is:

|  |  |
| --- | --- |
| 1  2  3 | constructor local variable  method local variable  this is instance variable |

Here, the first 2 calls are retrieving the local variables.

To access the instance variable from the local scope, we can use *this* keyword like it is shown in *printInstanceVariable()* method.

**2.2. Hiding and The Hierarchy**

Similarly, when both the child and the parent classes have a variable with the same name, the child’s variable hides the one from the parent.

Let’s suppose we have the parent class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class ParentVariable {        String instanceVariable = "parent variable";        public void printInstanceVariable() {          System.out.println(instanceVariable);      }  } |

After that we define a child class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class ChildVariable extends ParentVariable {        String instanceVariable = "child variable";        public void printInstanceVariable() {          System.out.println(instanceVariable);      }  } |

To test it, let’s initialize two instances. One with parent class and another with the child, then invoke the *printInstanceVariable()* methods on each of them:

|  |  |
| --- | --- |
| 1  2  3  4  5 | ParentVariable parentVariable = new ParentVariable();  ParentVariable childVariable = new ChildVariable();    parentVariable.printInstanceVariable();  childVariable.printInstanceVariable(); |

The output shows the property hiding:

|  |  |
| --- | --- |
| 1  2 | parent variable  child variable |

**In most cases, we should avoid creating variables with the same name both in parent and child classes**. Instead, we should use a proper access modifier like *private*and provide getter/setter methods for that purpose.

**3. Method Hiding**

Method hiding may happen in any hierarchy structure in java. When a child class defines a static method with the same signature as a static method in the parent class, then the child’s method *hides* the one in the parent class. To learn more about the *static* keyword,  [this write-up is a good place to start.](https://www.baeldung.com/spring-bean-scopes)

The same behavior involving the instance methods is called method overriding. To learn more about method overriding checkout our [guide here](https://www.baeldung.com/java-method-overload-override).

Now, let’s have a look at this practical example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class BaseMethodClass {        public static void printMessage() {          System.out.println("base static method");      }  } |

*BaseMethodClass* has a single *printMessage() static* method.

Next, let’s create a child class with the same signature as in the base class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class ChildMethodClass extends BaseMethodClass {        public static void printMessage() {          System.out.println("child static method");      }  } |

Here’s how it works:

|  |  |
| --- | --- |
| 1 | ChildMethodClass.printMessage(); |

The output after calling the *printMessage()* method:

|  |  |
| --- | --- |
| 1 | child static method |

The *ChildMethodClass.printMessage()*hides the method in *BaseMethodClass*.

**3.1. Method Hiding vs Overriding**

Hiding doesn’t work like overriding, because static methods are not polymorphic. Overriding occurs only with instance methods. It supports late binding, so which method will be called is determined at runtime.

**On the other hand, method hiding works with static ones. Therefore it’s determined at compile time**.

**4. Conclusion**

**In this article, we went over the concept of method and variable hiding in Java**. We showed different scenarios of variable hiding and shadowing. The important highlight of the article is also comparing the method overriding and hiding.