

# Technical Note No. 105

## Understanding Java Inheritance

Published: May 2, 2008. Last reviewed on September 22, 2008

By [Daryl Close](#), Professor of Computer Science and Philosophy, Heidelberg College, Tiffin, Ohio

**Summary:** This note explains the concept of Java inheritance.

Java inheritance must be stated very precisely. We know that constructors are not inherited. Class (static) fields and methods are not inherited. What, then, may be inherited?

A child class inherits all *accessible* instance fields and instance methods of the parent class. Note the emphasis on the term “accessible.” This is a source of much confusion in the Java textbook market where authors sometimes misstate Java inheritance as “all instance fields and methods,” or worse, “all fields and methods.”<sup>1</sup> This is confusing to the Java student because in good program design, the most common accessibility modifier for instance fields is private. But, private instance fields are *not* inherited, as is clearly stated by the creators of the Java language:

A class inherits from its direct superclass and direct superinterfaces all the non-private fields of the superclass and superinterfaces that are both accessible to code in the class and not hidden by a declaration in the class.

Note that a private field of a superclass might be accessible to a subclass (for example, if both classes are members of the same class). Nevertheless, a private field is never inherited by a subclass.<sup>2</sup>

The confusion here is compounded by the fact that inheritance brings about memory allocation for private instance fields as well as for those that have some level of visibility outside of the class. But, the student may rest assured that memory is also allocated for other non-inherited variables, e.g., static fields, so the concept of Java inheritance is not inconsistent at all, just limited to accessible instance fields and methods. If a derived class wants to access the private instance fields of its parent class, it must do so like everybody else, viz., through getter and setter methods—if any—in the parent class.

Although constructors aren’t inherited, inheritance affects the base class constructors. The key to understanding constructors in a derived class is knowing what happens when we create an object of a derived class type. Object creation via the new operator causes the derived class constructor to be called as usual. This in turn causes the base class default constructor to be called. The reason this second, implicit call to the base class constructor occurs is because the derived class needs to initialize any base class instance variables that it has inherited.

Normally, we don’t want this second step to occur behind the scenes. Instead, we want to have our derived class constructor call the base class constructor *explicitly*. This is done with `super()` or `super( parameter list )`. This invocation of a base class constructor must occur in the first line of the derived class default constructor. The invocation is chained to each ancestor class all the way to the Object class. This is called constructor chaining.

Finally, in Java, a child class may inherit from just one parent. Multiple inheritance of classes is not permitted in Java. A class may implement multiple interfaces, however.

---

<sup>1</sup> For example, “When one class inherits from another, it gets *all* the methods and fields from that parent class.” Guzdial, Mark, and Barbara Ericson. 2007. *Introduction to Computing & Programming with Java: A Multimedia Approach*. Upper Saddle River, NJ: Pearson Education, Inc., 348.

<sup>2</sup> Gosling, James, Bill Joy, and Guy Steele. [1996], 2000. “8.3 Field Declarations,” *Java Language Specification*. 2d ed. Santa Clara, CA: Sun Microsystems, Ch. 8.  
[http://java.sun.com/docs/books/jls/second\\_edition/html/classes.doc.html#21831](http://java.sun.com/docs/books/jls/second_edition/html/classes.doc.html#21831). (Accessed 14 April 2008).