

# 10.7 final Methods and Classes

We saw in Sections 6.3 and 6.10 that variables can be declared `final` to indicate that they cannot be modified *after* they’re initialized—such variables represent constant values. You also declare method parameters `final` to prevent them from being modified in the method’s body. It’s also possible to declare methods and classes with the `final` modifier.

## Final Methods Cannot Be Overridden

A **final method** in a superclass *cannot* be overridden in a subclass—this guarantees that the `final` method implementation will be used by all direct and indirect subclasses in the hierarchy. Methods that are declared `private` are implicitly `final`, because it’s not possible to override them in a subclass. Methods that are declared `static` are also implicitly `final`. A `final` method’s declaration can never change, so all subclasses use the same method implementation, and calls to `final` methods are resolved at compile time—this is known as **static binding**.

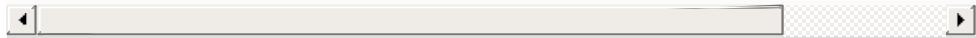
# Final Classes Cannot Be Superclasses

A **final class** cannot be extended to create a subclass. All methods in a **final** class are implicitly **final**. Class **String** is an example of a **final** class. If you were allowed to create a subclass of **String**, objects of that subclass could be used wherever **Strings** are expected. Since class **String** cannot be extended, programs that use **Strings** can rely on the functionality of **String** objects as specified in the Java API. Making the class **final** also prevents programmers from creating subclasses that might bypass security restrictions.

We've now discussed declaring variables, methods and classes **final**, and we've emphasized that if something *can* be **final** it *should* be **final**—this is another example of the *principle of least privilege*. When we study concurrency in [Chapter 23](#), you'll see that **final** variables make it much easier to parallelize your programs for use on today's multi-core processors. For more insights on the use of **final**, visit

---

<http://docs.oracle.com/javase/tutorial/java/IandI/fin>



## Common Programming

# Error 10.5

*Attempting to declare a subclass of a `final` class is a compilation error.*



## Software Engineering Observation 10.6

*In the Java API, the vast majority of classes are not declared `final`. This enables inheritance and polymorphism.*

*However, in some cases, it's important to declare classes `final`—typically for security reasons. Also, unless you carefully design a class for extension, you should declare the class as `final` to avoid (often subtle) errors.*



## Software Engineering Observation 10.7

*Though `final` classes cannot be extended, you can reuse them via composition.*