Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java : Final methods/5b1e5f467becc0459deae3b7

TUTORIAL

Java : Final methods

Occasionally, you want to prevent someone from forming a subclass from one of your classes. Classes that cannot be extended are called final classes, and you use the final modifier in the definition of the class to indicate this. For example, let us suppose we want to prevent others from subclassing the Executive class. Then, we simply declare the class by using the final modifier as follows:

```
final class Executive {
// ...
}
// The following class is illegal.
class B extends Executive { // ERROR! Can't subclass Executive
// ...
}
```

As the comments imply, it is illegal for B to inherit A since A is declared as final. While method overriding is one of Java's most powerful features, there will be times when you will want to prevent it from occurring. To disallow a method from being overridden, specify final as a modifier at the start of its declaration. Methods declared as final cannot be overridden. The following fragment illustrates final:

```
class A {
final void meth() {
System.out.println("This is a final method.");
}
}
class B extends A {
void meth() { // ERROR! Can't override.
System.out.println("Illegal!");
}
}
```

Because meth( ) is declared as final, it cannot be overridden in B. If you attempt to do so, a compile-time error will result.

Methods declared as final can sometimes provide a performance enhancement: The compiler is free to inline calls to them because it “knows” they will not be overridden by a subclass. When a small final method is called, often the Java compiler can copy the bytecode for the subroutine directly inline with the compiled code of the calling method, thus eliminating the costly overhead associated with a method call. Inlining is only an option with final methods. Normally, Java resolves calls to methods dynamically, at run time. This is called late binding. However, since final methods cannot be overridden, a call to one can be resolved at compile time. This is called early binding.

There is only one good reason to make a method or class final: to make sure that the semantics cannot be changed in a subclass. For example, the getTime and setTime methods of the Calendar class are final. This indicates that the designers of the Calendar class have taken over responsibility for the conversion between the Date class and the calendar state. No subclass should be allowed to mess up this arrangement. Similarly, the String class is a final class. That means nobody can define a subclass of String. In other words, if you have a String reference, then you know it refers to a String and nothing but a String.