# Overriding Methods: The Solution

The object oriented solution to this problem is to define a new class, call it SlowCar, which inherits from Car and imposes the additional constraint that a car may not go faster than 70 mph (112.65 kph).

To do this you'll need to adjust the two places that speed can be changed, the constructor and the accelerate() method. The constructor has a different name because all constructors are named after their classes but the accelerate() method must be overridden. This means the subclass has a method with the same signature as the method in the superclass.

public class SlowCar extends Car {  
  
 private static final double speedLimit = 112.65408; // kph == 70 mph  
  
 public SlowCar(String licensePlate, double speed, double maxSpeed,  
 String make, String model, int year, int numberOfPassengers, int numDoors) {  
   
 super(licensePlate, speed, maxSpeed, make, model, year,   
 numberOfPassengers, numDoors);  
 if (speed > speedLimit) this.speed = speedLimit;  
   
 }  
  
 public void accelerate(double deltaV) {  
  
 double speed = this.getSpeed() + deltaV;  
   
 if (speed > speedLimit) {  
 super.accelerate(speedLimit - this.getSpeed());  
 }  
 else {  
 super.accelerate(deltaV);   
 }   
   
 }  
   
}

The first thing to note about this class is what it doesn't have, getSpeed(), getLicensePlate(), getMaximumSpeed(), setLicensePlate() methods or speed, maxSpeed and numDoors fields. All of these are provided by the superclass Car. Nothing about them has changed so they don't need to be repeated here.

Next look at the accelerate() method. This is different than the accelerate() method in Car. It imposes the additional constraint. Since the speed and maxSpeed fields from Car are protected, they're accessible from this subclass.

The constructor is a little more complicated. First note that if you're going to use a non-default constructor, that is a constructor with arguments, you do need to write a constructor for the subclass, even if it's just going to do the exact same thing as the matching constructor in the superclass. You cannot simply inherit Car's constructor because that constructor is named Car() and this one must be named SlowCar().

The constructor needs to set the value of name, url, and description. However they're not accessible from the subclass. Instead they are set by calling the superclass's constructor using the keyword super. When super is used as a method in the first non-blank line of a constructor, it stands for the constructor of this class's superclass.

The immediate superclass's constructor *will* be called in the first non-blank line of the subclass's constructor. If you don't call it explicitly, then Java will call it for you with no arguments. It's a compile time error if the immediate superclass doesn't have a constructor with no arguments and you don't call a different constructor in the first line of the subclass's constructor.

The use of the ternary operator in the constructor call is unusual. However, it's necessary to meet the compiler's requirement that the invocation of super be the first line in the subclass constructor. Otherwise this could be written more clearly using only if-else.

[Previous](http://docs.google.com/11.html) | [Next](http://docs.google.com/13.html) | [Top](http://docs.google.com/index.html) | [Cafe au Lait](http://www.cafeaulait.org/)

Copyright 1997-2000, 2003 Elliotte Rusty Harold

[elharo@metalab.unc.edu](mailto:elharo@metalab.unc.edu)

Last Modified September 30, 2003