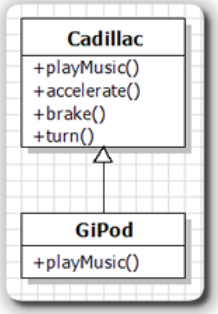
Contraction

Suppose you have a class which simulates a Cadillac. It has an exceptionally fine sound system, which required a lot of effort to implement and of which you're especially proud.

Now you want to reuse that sound system in a portable `GiPod` class: Because you've already created the `Cadillac` class, why not just create a derived class, and then eliminate all the member functions that have nothing to do with playing music, **transforming the car into a mere sound system**?



To reuse the code you've already written, you replace `brake()`, `accelerate()`, and all the other "extra" methods from the `Cadillac` class with empty braces. In traditional computer-science terms, you replace them with a **NOP** (**N**o **OP**eration).

This practice, called **contraction**, is a trap! You should avoid doing this for two reasons:

- You're **violating the substitutability rule**. You will undoubtedly break some code that relied on all `Cadillac` objects carrying out certain operations.
- It's **more work than doing the right thing!**

Let's look at how you can use **private inheritance** and **composition** to do this correctly.