Inheritance can be dangerous

```
public class X {
   public void do() { ... }
}

public class Y extends X {
   public void work() {
       do();
   }
}
```

```
public class X {
   public void do() {
       print(work());
    }
    private int work() {
       return 33;
    }
}
```

Class Y has very strong coupling with class X. If anything changes in superclass X , Y may not even compile. Suppose in future class X implements a method `work` with a different signature than the one of method `work` inside class Y.

When to prefer delegation

- **Preservation of clients' code:** Changing the interface of a back-end class (in delegation, the class that encapsulates an existing behavior) breaks less code than changing the interface of a superclass. In the former case, what is strictly needed is to change the implementation of the front-end class (in delegation, the class holding a reference to the back-end class), while in the latter case the interface of the front-end class must change as well (and thus clients' code)
- **Inherited behavior is seldom needed:** When using delegation, creation of back-end objects can be delayed (or even avoided, if they're not needed at all). If you use inheritance instead, you can't avoid the subclass object bringing along an image of a superclass object throughout its lifecycle

When to prefer inheritance

- **Multiple method calls**: in delegation, a method is just calling another method of another object with no other code surrounding this call. Moreover, the latter method may perform an additional call. When this process gets longer, code performances inevitably fall. If we used inheritance, there would be only one method call, regardless of how many steps far the method is in the hierarchy
- **Adding subclasses**: if you have a bit of code that relies only on a superclass interface, that code can work with a new subclass without change. This is not true of composition, unless you use composition with interfaces