Day 21: The facade design pattern

The facade design pattern is another kind of wrapper pattern, meant to simplify class usage for developers in the future. Simply provide a simplified interface as access to your base class! The adapter pattern is meant to make **existing** classes work together well, and the decorator pattern is meant to add functionality to classes — the facade design pattern is simply an ease-of-access simplification.

Example: say you just finished factoring out your components with the bridge pattern in mind. But now it's too hard to construct a usable object for people unfamiliar with your library - too many constructors, too many stacked traits. Here's where providing a facade "helper" class could help!

```
1  trait Component1 {
2    val toInit1: Int
3
4    def myInit(myData: Int): Int = toInit1 + myData
5  }
6
7  trait Component2 {
8    def myOtherInit(helpHere: Int): Int = helpHere * 4
9  }
10
11 class FacadeClass extends Component1 with Component2 {
12   override val toInit1: Int = 3
13
14   def facadeHelper(seed: Int): Int = myOtherInit(myInit(3))
15 }
```

Now the invoker of `FacadeClass` is using three components while only worrying about one bit of state. We could've asked them to initialize `Component1` and `Component2` separately, but we didn't.