Day 31: The mediator design pattern

A standard best practice is to focus your code design on discrete, well-bounded class components, rather than blurring the boundaries of different components together. But at some point, we need different components to interact, right? This is where the **mediator** design pattern comes in — *it defines interactions between discrete components*.

Arguably, any class that uses multiple discrete components is a mediator, but obviously there are good and bad mediators. Good design should yield mediators that are as constrained and abstracted as the component they themselves use. Let's take the book's example of a `School` mediator.

We have here one component for `Student` and one for `Group` — leaving these separate is good component-wise construction, but now we need their interactions. We define these in a mediator, per below:

```scala
case class Student(name: String, age: Int) {
  def notify(message: String): Unit = {
    println(s"Student $name notified: '$message'.")
  }
}

case class Group(name: String)

// the mediator
class School {
  val studentsToGroups: Map[Student, Set[Group]] = Map()
  val groupsToStudents: Map[Group, Set[Student]] = Map()

  def addStudentToGroup(student: Student, group: Group): Unit = {
    studentsToGroups.getOrElseUpdate(student, Set()) += group
    groupsToStudents.getOrElseUpdate(group, Set()) += student
  }

  def isStudentInGroup(student: Student, group: Group): Boolean = {
    groupsToStudents.getOrElse(group, Set()).contains(student) &&
studentsToGroups.getOrElse(student, Set()).contains(group)
  }

  def notifyStudentsInGroup(group: Group, message: String): Unit = {
    groupsToStudents.getOrElse(group, Set()).foreach(_.notify(message))
  }
}
```

A natural question to ask — why delegate responsibility of component interactions to a third component, instead of allowing a component a little more awareness of other components? Doesn't the third component introduce more complexity? I'd say no, for a few reasons:

1. Main problem: explosion of independent opinions. Say you have component A - either components B - Z all independently manage A, or mediator 1 focuses on composing A with B, and A with C, and ((A with B) with C) and so on. A mediator lets you centralize and standardize all those interactions.
2. Mediators let us stay focused on sharpening our components — how they're accessed, evolved, constructed — and letting the mediators stay focused on keeping well-defined contracts for component interaction.

Mediators are everywhere. Any implicit context, some constructor args, bad inheritence — it's all about introducing state into a component so that a component can mediate its own interactions with other things. Being more exacting about what a mediator is allows us to unearth common patterns