Day 29: Variance in Scala

Variance is a complex subject, but it powers literally everything you do in Scala, and whether you realize it or not, you probably have some expectations of variance when you code every day.

The basic problem of variance is this: given a box type `C[_]`, **if T1 <: T2, is C[T1] <: C[T2]?** If it is, we call that behavior **covariance** — you can treat a box of children (`C[T1]`) identically to a box of their parents (cool sentences!). If not, the behavior is invariant — we need to treat boxes of children differently than boxes of parents!

Here's an example:

```
// basic OOP hierarchy
trait BaseTrait
class Sub1 extends BaseTrait
class Sub2 extends BaseTrait

// this is an invariant box, and so can't be reliably used in ways that expect covariance!
case class InvariantBox[T <: BaseTrait](f1: T)
def takeInvariant(fs: InvariantBox[BaseTrait]): Unit = println(fs)

takeInvariant(InvariantBox[Sub1](new Sub1)) // boom
takeInvariant(InvariantBox(new Sub1)) // NO BOOM - Scala is a sneaky sonuvabitch
val ie = InvariantBox(new Sub1); takeInvariant(ie) // BUT THIS IS BOOM!
takeInvariant(InvariantBox[BaseTrait](new Sub1))

// now we've said "go ahead, treat Box[Parent] equally to Box[Child]"!
case class CovariantBox[+T <: BaseTrait](f1: T)
def takeCovariant(fs: CovariantBox[BaseTrait]): Unit = println(fs)

// now all work right away
takeCovariant(CovariantBox[Sub1](new Sub1))
takeCovariant(CovariantBox(new Sub1))
val ce = CovariantBox(new Sub1); takeCovariant(ce)
takeCovariant(CovariantBox[BaseTrait](new Sub1))
```

This all maybe seems a little academic - where does this really matter? Well, Lists! That's pretty important, right? The signature of List objects in Scala is the following:

```
sealed abstract class List[+A] extends ...
```

This is exactly identical to our case here! If `List` wasn't covariant on its generic, we wouldn't be able to do this (via the `apply` method):

```
val l = List(new Sub1, new Sub2, new Sub1, new BaseTrait {})
```

Another example is with functions, which are all instances of `FunctionN` and are contravariant on their arguments and covariant on their return types:

```
trait Function1[-T1, +R] extends AnyRef { ... }
```