What *may* make a covariant type unsound?

Generic parameter type may appear as the type of a method parameter.

```
class Queu[+T] {
  def append(x:T) =
  ...
}
```

Describe the Liskov Substitution Principle.

If T supports the same operations as U and all of T's operation have less strict requirements and provide more than the corresponding operations in U.

Then T is a subtype of U and you can substitute a T where a U is required.

Describe a fully persistent data structure.

Old versions remain available even after extensions or modifications.

Give an example of a good candidate for contravariance.

```
trait OutputChannel[-T] {
  def write(x:T)
}
```

It makes sense that `OutputChannel` of `AnyRef` is a subtype of `OutputChannel` of `String` because a class that can write out an `AnyRef` can certainly write out `Strings`. The converse is not true.

Give an example of a good candidate for covariance.

A functional `Queue`.

```
class Queue[+T] {
  dep append[U>:T] (x:U) :
    Queue[U] = ...
}
```

This allows the adding of `Queue[Apple]` and an `Orange` to make a new `Queue[Fruit]`.

Why are Java arrays covariant?

Before Java introduced generics, it was the only way to support method signatures that operated on all types of arrays.

```
void sort(Object[]a, Comparator cmp)
```

Interact with Java covariant arrays using Scala nonvariant `Arrays`.

Scala lets you cast an array of T to an array of any supertype of T.

Describe variance annotations.

They appear before formal parameter.

```
MyClass[-T] contravariant
MyClass[T] nonvariant
MyClass[+T] covariant
```

What are the downsides of Java arrays being covariant?

- Efficiency, type of elements added may be checked against store type.
- Storage is not covariant, resulting in runtime `ArrayStoreExceptions` the compiler cannot catch.

Modify visibility of primary constructor.

```
class MyClass private (
  val param1: Int
  val param2: String
)
```

What's a good use of object private fields?

Escaping the variance checker's usual prohibition on reassignable fields of a covariant parameter.

What does the compiler guarantee regarding variance annotations and usage?

How so?

Correctness, by classifying all positions in a class as positive, negative, or neutral.

Nonvariants can be used at all three, covariants at positive only, and contravariants at negative only.

Why doesn't mutable state mesh well with covariance?

If the var is of the covariant type generated, setter and getter have methods with the covariant parameter.