

Variance Positions



Variance

Variance question: if A extends B, should Thing<A> "extend" Thing?

- yes – covariant
- no – invariant
- hell no, quite the contrary – contravariant

```
trait List<out A> // covariant
trait Semigroup<A> // invariant
trait Vet<in A> // contravariant
```

Rule of thumb: how to determine variance

- if the type produces (outputs) values of type A, then covariant
- if the type consumes (takes input) values of type A, then contravariant
- otherwise, invariant

Variance Positions

Val & var properties are in "out" (= covariant) position

```
class Vet<in A>(val favoriteAnimal: A) // error: "in" type A is in "out" position
```

Vars are *also* in "in" (= contravariant) position => must be *invariant*

```
class MOption<out A>(var contents: A) // error: "out" type A is in "in" position
```

Method argument types are in "in" (= contravariant) position

```
class LList<out A> {  
  // error: "out" type A is in "in" position  
  def add(element: A): LList <A> = ???  
}
```

Method return types are in "out" (covariant) position

```
abstract class Vet<in A> {  
  // error: "in" type A is in "out" position  
  def rescueAnimal(): A  
}
```

Overcoming Problems

Method args & covariance: widen the type

```
abstract class LList<out A>
def <B, A:B> LList<A>.add(element: B): LList<B>
```

Method return types & contravariance: narrow the type

```
class RepairShop<in A: Vehicle> {
  def <B : A> repair(vehicle: B): B = vehicle
}
```

Kotlin rocks

