In [1]:
```
class TypeClass[T] //in java - class GenericType<T>
```
```
defined class TypeClass
```

In [4]:
```
//Allow any TypeClass treated as Child when a ChildClass is used
//Default INVARIANCE
class ParentClass
class ChildClass extends ParentClass

class TypeClass[T]

val p = new TypeClass[ParentClass]
val c = new TypeClass[ChildClass]
```
```
defined class ParentClass
defined class ChildClass
defined class TypeClass
p: $user.TypeClass[$user.ParentClass] = cmd3$$user$TypeClass@75ceeb77
c: $user.TypeClass[$user.ChildClass] = cmd3$$user$TypeClass@15b91be8
```

In [4]:
```
val pr: TypeClass[ParentClass]=c //generic polymorphism
//failed because p n c are completely dif variables, does not have any re
```
```
Main.scala:30: type mismatch;
 found   : cmd4.this.$ref$cmd3.TypeClass[cmd4.this.$ref$cmd3.ChildClas
s]
 required: cmd4.this.$ref$cmd3.TypeClass[cmd4.this.$ref$cmd3.ParentCla
ss]
Note: cmd4.this.$ref$cmd3.ChildClass <: cmd4.this.$ref$cmd3.ParentClas
s, but class TypeClass is invariant in type T.
You may wish to define T as +T instead. (SLS 4.5)
}.apply
  ^
```

In [5]:
```
//Co-variance
class TypeClass[+T]

val p = new TypeClass[ParentClass]
val c = new TypeClass[ChildClass]

val pr:TypeClass[ParentClass]=c

// p n c inherits the same behavior
//due to + sign both p n c are related now. Passing a child to a parent w
```

```
defined class TypeClass
p: $user.TypeClass[ParentClass] = cmd4$$user$TypeClass@67699a77
c: $user.TypeClass[ChildClass] = cmd4$$user$TypeClass@4b9f0034
pr: $user.TypeClass[ParentClass] = cmd4$$user$TypeClass@4b9f0034
```

In [6]:
```
//CONTRA-VARIANCE - reverse of co-variance
class TypeClass[-T]

val p = new TypeClass[ParentClass]
val c = new TypeClass[ChildClass]

val pr:TypeClass[ChildClass]=p
//usually this is not possible usually - not allowed in Java
//accessing a child class type with a parent class type
```

```
defined class TypeClass
p: $user.TypeClass[ParentClass] = cmd5$$user$TypeClass@117240d6
c: $user.TypeClass[ChildClass] = cmd5$$user$TypeClass@2ed98764
pr: $user.TypeClass[ChildClass] = cmd5$$user$TypeClass@117240d6
```

In [10]:
```
//LOWER, UPPER BOUNDS
//Hint: UML inheritance direction

class SomeOtherClass
class GreatList[T] {
  def add[S>:T](n:S)={ //LowerBound S must be Super Class of T //S must b
    //> UML inheritance direction
    //giong from top to down

  }
  def altAdd[S<:T](n:S)={ //UpperBound S must be a sub class of T //S mus

  }
  //works with traits too
  def altTraitAdd[T<:Ordered[T]](n:T)= { //T must be mixed with ordered T
    //anything passed to this must be a OrederedTrait
    // Ordered is a trait

  }


  class SomeList[T<:Ordered[T]]

//   val c=new SomeList[ParentClass] //=>Error


  class OrderedParentClass extends Ordered[OrderedParentClass]{
    def compare(that:OrderedParentClass)= {
      0
    }
  }
  val c = new SomeList[OrderedParentClass]
  val g = new GreatList[OrderedParentClass]
  val p = new OrderedParentClass()
  g.altTraitAdd(p)
}
```

```
defined class SomeOtherClass
defined class GreatList
```

In [ ]:
```
  class SomeNiceClass extends Ordered[SomeNiceClass]{

  }
```