# Exercise 3:

Implement a generic Tree library. A tree is parameterized by a comparable trait and must implement methods to find an element (returning a boolean) and compute the maximum element of the tree using a 'max' method. A Tree can be either a Node or a Leaf

```
trait Comparable[T] {
  def compareTo(o: T): Int
}
trait Tree[T <: Comparable[T]] {
  def find(x: T): Boolean
  def max: T
}
```

dcc
CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

# Exercise 3:

```
trait Comparable[T] {
  def compareTo(o: T): Int
}
trait Tree[T <: Comparable[T]] {
  def find(x: T): Boolean
  def max: T
}
```

```
class Node[T <: Comparable[T]](
    val value: T,
    val left: Tree[T],
    val right: Tree[T]
) extends Tree[T] {
  def max: T = {
    val innerMax =
      if (left.max.compareTo(right.max) > 0) left.max
      else right.max
    if (value.compareTo(innerMax) > 0) value else innerMax
  }
  def find(x: T): Boolean =
    (value.compareTo(x) == 0) || left.find(x) || right.find(x)
}

class Leaf[T <: Comparable[T]](val value: T) extends Tree[T] {
  def max: T = value
  def find(x: T): Boolean = (value.compareTo(x) == 0)
}
```

# Exercise 3:

Write a use example:

```
class Person(val name: String, val age: Int) extends Comparable[Person] {
  override def compareTo(o: Person): Int = age.compareTo(o.age)
  override def toString() = s"Person($name, $age)"}

val tree: Tree[Person] = new Node[Person](
  new Person("Juan", 20),
  new Leaf[Person](new Person("Pedro", 30)),
  new Leaf[Person](new Person("Maria", 40))
)

println(tree.max)
println(tree.find(new Person("Pedro", 30)))
println(tree.find(new Person("Pedro", 31)))
```

```
Person(Maria, 40)
true
false
```