Dr. Th. Letschert
Softwarekomponenten

THM TECHNISCHE HOCHSCHULE MITTELHESSEN
MNI Mathematik, Naturwissenschaften und Informatik

Aufgabenblatt 4

Aufgabe 1

Machen den Typ der Binärbäume:

```
enum Tree[+A] {
  case Node(left: Tree[A], right: Tree[A])
  case Leaf(value: A)
}
```

zum Funktor:

```
trait Functor[F[_]] {
  extension[A, B] (fa: F[A]) {
    def map(f: A => B): F[B]
  }
}

given Functor[Tree] with {
  ???
}

val tree_1 = Node(Leaf(1), Node(Node(Leaf(2), Leaf(3)), Leaf(4)))
val tree_2 = tree_1.map( _*3 )
```

Aufgabe 2

Die (unvermeidliche) Typklasse *Showable* kann definiert werden als

```
trait Showable[A] {
  def show(a: A): String
}
object Showable {
  def apply[A: Showable] = summon[Showable[A]]
}
```

1. Natürlich ist *String* und *Int* *Showable*:

   ```
   given Showable[String] with {
     ...
   }

   given Showable[Int] with {
     ...
   }
   ```

2. Ein Paar von A-Werten ist definiert als:

```
case class Pair[A](x: A, y: A)
```

   Wenn *A Showable* ist, dann ist auch *Pair[A] Showable*:

```
given [T](using showT: Showable[T]): Showable[Pair[T]] with {
  ... // // Ergänzen Sie
}

val p1 = Pair[String]("Hallo", "Welt")
val p2 = Pair(47, 11)

// Test
val p1AsString: String = ... p1 ... // p1 als String
val p2AsString: String = ... p2 ... // p2 als String
```

3. Das geht natürlich auch mit Listen:

```
...

val l1 = List("Hallo", "Welt")
val l2 = List(47, 11)

val str1: String = ... l1 ... // l1 als String
val str2: String = ... l2 ... // l2 als String
```

4. Wenn eine List mit Elementen, die *Showable* sind, in einen String transformiert werden können, dann kann jeder Funktor F der in eine Liste transformiert werden kann, auch in einen String transformiert werden:

```
trait Functor[F[_]] {
  extension[A, B] (fa: F[A]) {
    def map(f: A => B): F[B]
  }
}

trait Listable[F[_]: Functor] {
  def toList[A](fa: F[A]) : List[A]
}

given [F[_], T](using listableF: Listable[F], showT: Showable[T]):
    Showable[F[T]] with {
  def show(ft: F[T]): String = ...
}

// Test:
enum Tree[+A] {
  case Node(left: Tree[A], right: Tree[A])
  case Leaf(value: A)
}
import Tree._

given Functor[Tree] with {
  ...
}

// z.B. eine Infix-Darstellung des Baums
given Listable[Tree] with {
  ...
```

```
}

val tree = Node(Leaf(1), Node(Node(Leaf(2), Leaf(3)), Leaf(4)))
val str: String = Showable[Tree[Int]].show(tree) // [[1,2,3,4]]
```

5. Welche Ergänzungen sind notwendig, wenn jetzt ein Baum von Paaren zu einem String transformiert werden soll:

```
...
val tree = Node(Leaf(Pair(1, 2)), Node(Node(Leaf(Pair(3,4)),
    Leaf(Pair(5,6))), Leaf(Pair(7,8))))
val str: String = Showable[Tree[Pair[Int]]].show(tree)
```

6. Ein Contrafunktor ist etwas, vor das man mit *contraMap* eine Operation setzen kann. *Showable* ist ein Contrafunktor.

```
trait ContraFunctor[F[_]] {
  extension[A, B] (fb: F[B]) {
    def contraMap(f: A => B): F[A]
  }
}

given ContraFunctor[Showable] with {
  ...
}
```

   Wenn es eine Funktion gibt, die einen Typ A in etwas von Typ B transformiert, und B *Showable* ist, dann dann kann ein *a: A* in einen String transformiert werden. Formulieren Sie dies in Scala.

Aufgabe 3

Ein noch etwas frischer Adept der funktionalen Programmierung definiert den Typ *ToInt[A]* und behauptet dann, dass dieser Typ zu einem Funktor gemacht werden kann. Als Beweis wird folgendes Codefragment vorgelegt:

```
trait Functor[F[_]] {
  extension[A, B] (fa: F[A]) {
    def map(f: A => B): F[B]
  }
}

case class ToInt[A](fa: A => Int) {
  def apply(a: A) = fa(a)
}

given Functor[ToInt] with {
  extension[A, B] (fa: A => Int) {
    def map(f: A => B): B => Int = b => 4711
  }
}
```

Auf ihren kritischen Blick und eine leicht gemurmelte Erwähnung des Begriffs *Gesetze* hin, liefert er noch einen Test der Assoziativität:

```
def f(str: String): Int = str.length
def g(i: Int): Char = i.toString.charAt(0)

val x: ToInt[String] = ToInt( (str: String) => str.length )
```

```
val xMap_f: ToInt[Int] = x.map(f)
val xMap_f_Map_g: ToInt[Int] = xMap_f.map(g)

val xMap_fg: ToInt[Int] = x.map(f andThen g)

assert( xMap_f_Map_g(42) == xMap_fg(42) )
```

der glänzend bestanden wird.

Warum sind Sie immer noch nicht zufrieden?

Aufgabe 4

Set und *Map* der Scala-API haben (eventuell mehr als) eine *map*-Methode. Sind sie damit Funktoren?

Bedenken Sie dabei auch dass der *equals*-Methode bei Abbildungs- und Mengen-Kollektionen eine besondere Bedeutung zukommt.