Tuples and Generic Methods

Sorting Lists Faster

A good algorithm for this is merge sort. The idea is as follows:

- If the list consists of zero or one elements, it is already sorted
- Otherwise:
    - Separate the lists into two sub-lists, each containing around half of the elements of the original list
    - Sort the two sub-lists
    - Merge the two sorted sub-lists into a single sorted list

The SplitAt Function

- The ***splitAt*** function on lists returns two sublists
    - The elements up to the given index
    - The elements from that index
- The lists are returned in a pair

Pair and Tuples

- The pair consisting of x and y is written (x, y) in Scala
- Example: ***val pair = ("answer", 42)***
- The type of pair above is ***(String, Int)***
- Pairs can be also used as patterns: ***val (label, value) = pair***
- This works analogously for tuples with more than two elements

Translation of Tuples

- For a small n(up to 22), the tuple type (T_1, … T_n) is an abbreviation of the parameterized type: ***scala.Tuplen[T_1, … T_n]***
- A tuple expression (e_1, … e_n) is equivalent to the function application: ***scala.Tuplen(e_1, … e_n)***
- A tuple pattern (p_1, … p_n) is equivalent to the constructor pattern: ***scala.Tuplen(p_1, … p_n)***
- There is also a class TupleXXL that handles Tuples larger than 22 elements

The Tuple Class

- Here, all Tuplen classes are modeled after the following pattern:

```
case class Tuple2[T1, T2](_1: +T1, _2: +T2):
  override def toString = "(" + _1 + "," + _2 + ")"
```

- The fields of a tuple can be accessed with names _1, _2, …
- So instead of the pattern binding val (label, value) = pair, one could also have written:
  val label = pair._1
  val value = pair._2
- But the pattern matching form is generally preferred