△ nus-cs2030s-2122-s2 / lab7-beetee17 (Private)

```
Actions
                                                          Projects
<> Code
            Issues
                         Pull requests
                                                                          Security
               lab7-beetee17 / cs2030s / fp / InfiniteList.java
                                                                  Go to file
  master •
               / <> Jump to ▼
    beetee17 Cumulative submission
                                    Latest commit e713e55 9 days ago (1) History
A 2 contributors
                                                                      רֹ
307 lines (269 sloc) 8.31 KB
                                                   Raw
                                                          Blame
  1
      package cs2030s.fp;
  2
      import java.util.ArrayList;
  3
      import java.util.List;
  4
      import java.util.NoSuchElementException;
  5
  6
  7
      /**
       * Infinite List is similar to a Stream. It memoizes values and is lazyi
  8
  9
       * in evaluation.
 10
       * @author Brandon (Group 12A)
 11
 12
 13
      public class InfiniteList<T> {
 14
        private final Lazy<Maybe<T>> head;
 15
        private final Lazy<InfiniteList<T>> tail;
 16
 17
        private static final InfiniteList<?> SENTINEL = new Sentinel();
 18
 19
        /**
         * A private constructor. We only allow initialisation via
 20
         * generate and iterate.
 21
 22
 23
         */
 24
        private InfiniteList() {
          this.head = null;
 25
 26
          this.tail = null;
```

}

27

```
28
29
        * A private constructor. We only allow initialisation via
30
        * generate and iterate.
31
32
33
        * Oparam head The given head to be wrapped in Lazy.
34
        * Oparam tail The given tail to be wrapped in Lazy.
        */
35
       private InfiniteList(T head, Producer<InfiniteList<T>> tail) {
36
37
         this.head = Lazy.of(Maybe.some(head));
         this.tail = Lazy.of(() -> tail.produce());
38
       }
39
40
41
       /**
42
        * A private constructor. We only allow initialisation via
43
        * generate and iterate.
44
45
        * Oparam head The given head to.
        * Oparam tail The given tail to.
46
47
48
       private InfiniteList(Lazy<Maybe<T>> head, Lazy<InfiniteList<T>> tail) {
49
         this.head = head;
        this.tail = tail;
50
       }
51
52
53
       /**
54
       * Creates an InfiniteList according to a Producer.
55
        * @param <T> The type of the desired InfiniteList.
56
        * Oparam producer The given producer to generate values
57
        * @return An InfiniteList of generated values.
58
59
        */
       public static <T> InfiniteList<T> generate(Producer<T> producer) {
60
         return new InfiniteList<>(Lazy.of(() -> Maybe.some(producer.produce())),
61
62
             Lazy.of(() -> generate(producer)));
       }
63
64
65
66
       * Creates an InfiniteList according to an initial value and an iterator.
67
        * @param <T> The type of the desired InfiniteList.
68
        * @param seed The first value of the InfiniteList
69
70
        * Oparam next The given iterator to generate values
        * @return An InfiniteList of iterated values.
71
72
        */
```

```
73
        public static <T> InfiniteList<T> iterate(T seed, Transformer<T, T> next) {
 74
          return new InfiniteList<>(seed, () -> iterate(next.transform(seed), next)
 75
        }
 76
 77
        /** Returns the head's value of the InfiniteList.
 78
 79
         * Oreturn The head's value.
         */
 80
 81
        public T head() throws NoSuchElementException {
          return this.head.get().orElseGet(() -> this.tail.get().head());
 82
 83
        }
 84
 85
        /** Returns the tail of the InfiniteList.
 86
         * @return The tail of the InfiniteList..
 87
 88
 89
        public InfiniteList<T> tail() throws NoSuchElementException {
 90
          InfiniteList<T> tempTail = this.tail.get();
          if (this.head.get().equals(Maybe.none())) {
 91
            return tempTail.isSentinel() ? sentinel() : tempTail.tail();
 92
 93
          }
 94
          return tempTail;
        }
 95
 96
 97
        /**
         * Lazily applies the given transformation to each element in the list
 98
 99
         * and returns the resulting `InfiniteList`.
100
         * Oparam <R> type parameter for resulting `InfiniteList`
101
         * @param mapper the mapping function to be applied on the list.
102
103
104
         * @return an `InfiniteList` with its elements mapped.
105
         */
        public <R> InfiniteList<R> map(Transformer<? super T, ? extends R> mapper)
106
107
          return new InfiniteList<>(Lazy.of(() -> Maybe.some(mapper.transform(this.
108
              Lazy.of(() -> this.tail().map(mapper)));
109
        }
110
111
        /**
112
         * Lazily filters out elements in the list that fail a given
         * BooleanCondition. Marks removed elements as Maybe.none().
113
114
115
         * @param predicate The BooleanCondition to test values with.
         * @return The filtered InfiniteList.
116
117
         */
```

```
118
        public InfiniteList<T> filter(BooleanCondition<? super T> predicate) {
119
          Lazy<Maybe<T>> tempHead = Lazy.of(() -> predicate.test(this.head())
120
              ? Maybe.some(this.head())
121
              : Maybe.none());
122
          return new InfiniteList<>(tempHead, Lazy.of(() -> this.tail().filter(pred
123
        }
124
        /** Returns a Sentinel.
125
126
127
         * @param <T> The type of the desired Sentinel.
128
         * @return A Sentinel of the specified type.
129
         */
130
        public static <T> InfiniteList<T> sentinel() {
131
          @SuppressWarnings("unchecked")
          InfiniteList<T> temp = (InfiniteList<T>) SENTINEL;
132
133
          return temp;
134
        }
135
136
        /** Terminates an infinite list into a finite one with at most n elements.
137
138
         * Oparam n The maximum length of the truncated list.
         * @return The truncated list.
139
140
         */
        public InfiniteList<T> limit(long n) {
141
142
          if (n <= 0) {
143
            return sentinel();
144
145
146
          Producer<InfiniteList<T>> newTail = () -> this.head.get()
147
              .map(x -> this.tail.get().limit(n - 1))
              .orElseGet(() -> this.tail.get().limit(n));
148
149
150
          return new InfiniteList<>(this.head, Lazy.of(newTail));
        }
151
152
153
        /**
        * Truncates the list as soon as an element does not satisfy a predicate.
154
155
156
         * Oparam predicate The BooleanCondition to test elements with
157
         * @return The truncated list.
158
        public InfiniteList<T> takeWhile(BooleanCondition<? super T> predicate) {
159
160
          Lazy<Boolean> cond = Lazy.of(() -> this.head()).filter(predicate);
161
162
          Lazy<Maybe<T>> newHead = Lazy.of(() -> cond.get()
```

```
163
              ? Maybe.some(this.head())
164
              : Maybe.none());
165
166
          return new InfiniteList<T>(
167
              newHead,
168
              Lazy.of(() -> cond.get() && predicate.test(this.tail().head())
169
                ? this.tail().helper(predicate)
                : sentinel())
170
              );
171
172
        }
173
174
        /**
175
        * Helper function for the takeWhile method.
176
         * @param p The BooleanCondition to test elements with
177
         * @return An InfiniteList.
178
179
180
        private InfiniteList<T> helper(BooleanCondition<? super T> p) {
          return new InfiniteList<T>(this.head(),
181
182
              () -> p.test(this.tail().head())
183
              ? this.tail().helper(p)
184
              : sentinel());
        }
185
186
187
        /**
         * Checks if this is an instance of Sentinel.
188
189
190
         * @return true if the list is an instance of Sentinel, and false otherwis
191
         */
        public boolean isSentinel() {
192
193
          return false;
194
        }
195
196
197
         * Reduces the list into a single value.
198
199
         * @param <U> The type of the return value
200
         * Oparam identity The initial value
201
         * Oparam accumulator Combiner to combine two values
202
         * Oreturn The result of accumulating the list from right to left.
203
        public <U> U reduce(U identity, Combiner<U, ? super T, U> accumulator) {
204
205
          return accumulator.combine(this.tail().reduce(identity, accumulator), thi
206
        }
207
```

```
208
        /**
209
         * Gets the length of the list.
210
211
        * @return The length of the list.
212
        */
213
        public long count() {
214
          long v = this.head.get().equals(Maybe.none()) ? 0L : 1L;
215
          return v + this.tail.get().count();
216
        }
217
218
219
        * Collects the elements into a List.
220
        * @return A list of elements of the InfiniteList.
221
         */
222
223
        public List<T> toList() {
          List<T> ls = new ArrayList<>();
224
225
          ls.add(this.head());
226
          ls.addAll(this.tail().toList());
227
          return ls;
228
        }
229
230
        * Returns the String representation of the list.
231
232
233
        * @return Wraps the head and tail in square brackets.
234
        */
235
        public String toString() {
          return "[" + this.head + " " + this.tail + "]";
236
237
        }
238
239
        /**
240
        * Sentinel represents a list that contains nothing.
241
        * It is used to mark the end of the list.
242
243
        private static class Sentinel extends InfiniteList<Object> {
244
245
          /**
246
           * Constructor for a Sentinel instance.
247
          */
248
          Sentinel() {
249
            super();
250
          }
251
          @Override
252
```

```
253
          public Object head() throws NoSuchElementException {
254
            throw new java.util.NoSuchElementException();
255
          }
256
257
          @Override
258
          public InfiniteList<Object> tail() throws NoSuchElementException {
259
            throw new java.util.NoSuchElementException();
260
261
262
          @Override
263
          public <R> InfiniteList<R> map(Transformer<? super Object, ? extends R> m
            return InfiniteList.sentinel();
264
265
          }
266
267
          @Override
          public InfiniteList<Object> filter(BooleanCondition<? super Object> predi
268
269
            return InfiniteList.sentinel();
270
          }
271
          @Override
272
273
          public InfiniteList<Object> limit(long n) {
274
            return InfiniteList.sentinel();
275
          }
276
277
          @Override
          public List<Object> toList() {
278
279
            return List.of();
280
          }
281
282
          @Override
283
          public InfiniteList<Object> takeWhile(BooleanCondition<? super Object> pr
284
            return InfiniteList.sentinel();
285
          }
286
287
          @Override
          public <U> U reduce(U identity, Combiner<U, ? super Object, U> accumulato
288
289
            return identity;
290
          }
291
292
          @Override
          public long count() {
293
294
            return 0;
295
          }
296
297
          @Override
```

```
298
          public boolean isSentinel() {
299
            return true;
300
          }
301
302
          @Override
303
          public String toString() {
            return "-";
304
305
306
        }
307
      }
```