

[nus-cs2030s-2122-s2 / lab7-beetee17](#) Private

<> Code

Issues

Pull requests

Actions

Projects

Security

master ▾

[lab7-beetee17](#) / [cs2030s](#) / [fp](#) / [Maybe.java](#) /

Go to file

...

<> Jump to ▾



beetee17 Cumulative submission

Latest commit e713e55 9 days ago

History

1 contributor

192 lines (152 sloc) | 4.07 KB

Raw

Blame



```
1 package cs2030s.fp;
2
3 import java.util.NoSuchElementException;
4 import java.util.function.Consumer;
5
6 /**
7  * CS2030S Lab 5.
8  * AY21/22 Semester 2
9  *
10 * @author Brandon (Group 12A)
11 */
12 public abstract class Maybe<T> {
13     private Maybe() {
14     }
15
16     public static <T> Maybe<T> of(T t) {
17         if (t == null) {
18             return none();
19         } else {
20             return some(t);
21         }
22     }
23
24     public static <T> Maybe<T> none() {
25         @SuppressWarnings("unchecked")
26         Maybe<T> none = (Maybe<T>) None.NONE;
27         return none;
```

```
28     }
29
30     public static <T> Maybe<T> some(T t) {
31         return new Some<T>(t);
32     }
33
34     protected abstract T get() throws NoSuchElementException;
35
36     public abstract Maybe<T> filter(BooleanCondition<? super T> cond);
37
38     public abstract <U> Maybe<U> map(Transformer<? super T, ? extends U> transf
39
40     public abstract <U> Maybe<U> flatMap(Transformer<? super T,
41         ? extends Maybe<? extends U>> transformer);
42
43     public abstract T orElse(T t);
44
45     public abstract T orElseGet(Producer<? extends T> producer);
46
47     /**
48      * If the value within this Maybe is missing, do nothing.
49      * Otherwise, consume the value with the given consumer.
50      *
51      * @param consumer The consumer to consume the value
52      */
53     public abstract void consumeWith(Consumer<? super T> consumer);
54
55     private static class None extends Maybe<Object> {
56
57         private None() {
58             super();
59         }
60
61         private static final Maybe<?> NONE = new None();
62
63         @Override
64         protected Object get() throws NoSuchElementException {
65             throw new NoSuchElementException();
66         }
67
68         @Override
69         public Maybe<Object> filter(BooleanCondition<? super Object> cond) {
70             return Maybe.none();
71         }
72     }
```

```
73     @Override
74     public <U> Maybe<U> map(Transformer<? super Object, ? extends U> transformer) {
75         return Maybe.none();
76     }
77
78     @Override
79     public <U> Maybe<U> flatMap(Transformer<? super Object,
80         ? extends Maybe<? extends U>> transformer) {
81         return Maybe.none();
82     }
83
84     @Override
85     public Object orElse(Object o) {
86         return o;
87     }
88
89     @Override
90     public Object orElseGet(Producer<? extends Object> producer) {
91         return producer.produce();
92     }
93
94     @Override
95     public void consumeWith(Consumer<? super Object> consumer) {
96     }
97
98     @Override
99     public String toString() {
100         return "[]";
101     }
102
103     @Override
104     public boolean equals(Object o) {
105         if (this == o) {
106             return true;
107         }
108
109         if (o == null || !(o instanceof None)) {
110             return false;
111         }
112
113         return true;
114     }
115 }
116
117 private static final class Some<T> extends Maybe<T> {
```

```
118     private T t;
119
120     protected Some(T t) {
121         super();
122         this.t = t;
123     }
124
125     @Override
126     protected T get() throws NoSuchElementException {
127         return this.t;
128     }
129
130     @Override
131     public Maybe<T> filter(BooleanCondition<? super T> cond) {
132         if (this.t != null && !cond.test(this.t)) {
133             return Maybe.none();
134         } else {
135             return this;
136         }
137     }
138
139     @Override
140     public <U> Maybe<U> map(Transformer<? super T, ? extends U> transformer)
141         return Maybe.some(transformer.transform(this.t));
142     }
143
144     @Override
145     public <U> Maybe<U> flatMap(Transformer<? super T, ? extends Maybe<? exte
146         @SuppressWarnings("unchecked")
147         Maybe<U> maybeU = (Maybe<U>) transformer.transform(this.t);
148         return maybeU;
149     }
150
151     @Override
152     public T orElse(T t) {
153         return this.t;
154     }
155
156     @Override
157     public T orElseGet(Producer<? extends T> producer) {
158         return this.t;
159     }
160
161     @Override
162     public void consumeWith(Consumer<? super T> consumer) {
```

```
163         consumer.accept(this.t);
164     }
165
166     @Override
167     public String toString() {
168         return String.format("[%s]", this.t);
169     }
170
171     @Override
172     public boolean equals(Object o) {
173         if (this == o) {
174             return true;
175         }
176
177         if (o == null || !(o instanceof Some<?>)) {
178             return false;
179         }
180
181         Some<?> other = (Some<?>) o;
182
183         if (this.t == null && other.t == null) {
184             return true;
185         } else if (this.t == null) {
186             return false;
187         } else {
188             return this.t.equals(other.t);
189         }
190     }
191 }
192 }
```