

beetee17 / NUS_Y1S2 Public

<> Code

Issues

Pull requests

Actions

Projects

Wiki

!

main ▾

NUS_Y1S2 / CS2030S / Past Year Practices / PE2
/ pe2-2021s2 / cs2030s / fp / Try (solution).java /
<> Jump to ▾

Go to file

...



beetee17 Update 8 Apr

Latest commit 0b0d5e6 1 minute ago

History

1 contributor

177 lines (151 sloc) | 4.79 KB

Raw

Blame



```
1 package cs2030s.fp;
2
3 public abstract class Try<T> {
4     /**
5      * Return a failed Try with a given Throwable.
6      *
7      * @param <T> The type of the value contained in this Throwable.
8      * @param a The Throwable to initialize this failure with.
9      *
10     * @return The new failure.
11     */
12     public static <T> Try<T> failure(Throwable a) {
13         return new Failure<T>(a);
14     }
15
16     /**
17      * Return a success Try with the given value.
18      *
19      * @param <T> The type of the value contained in this Throwable.
20      * @param value The Throwable to initialize this success with.
21      *
22     */
23     public static <T> Try<T> success(T value) {
24         return new Success<T>(value);
25     }
26 }
```

```
27  /**
28   * Return a Try after running the producer. If the producer produces
29   * without error, return a success with the value produced. Else,
30   * return a failure with the error/exception.
31   *
32   * @param <T> The type of the value contained in this Throwable.
33   * @param producer The producer to initialize this Try with.
34   *
35   * @return The new Try.
36   */
37  public static <T> Try<T> of(Producer<? extends T> producer) {
38      try {
39          return success(producer.produce());
40      } catch (Throwable e) {
41          return failure(e);
42      }
43  }
44
45  public abstract T get() throws Throwable;
46
47  public abstract <R> Try<R> map(Transformer<? super T, ? extends R> mapper);
48
49  public abstract <R> Try<R> flatMap(Transformer<? super T, ? extends Try<? e
50
51  /**
52   * If the calling Try is a failure, return itself after running the consume
53   * on the throwabale; otherwise if the calling Try is a success, just retur
54   * it self. If the consumer fails the throwable is replaced with the new
55   * error.
56   *
57   * @param consumer The consumer to consume the throwable with.
58   *
59   * @return The new Try
60   */
61  public abstract Try<T> onFailure(Consumer<? super Throwable> consumer);
62
63  /**
64   * If the calling Try is a failure, return a success Try after running the
65   * given Transformer on the value; if the calling Try is a success, return
66   * itself without running the Transformer.
67   * If the transformer fails the throwable is replaced with the new
68   * error.
69   *
70   * @param transformer The transformer to transform the value with.
71   *
```

```
72     * @return The new Try
73     */
74     public abstract Try<T> recover(Transformer<? super Throwable, ? extends T>
75
76     private static class Success<T> extends Try<T> {
77         T value;
78
79         Success(T value) {
80             this.value = value;
81         }
82
83         public String toString() {
84             return "Success: " + String.valueOf(value);
85         }
86
87         public <R> Try<R> map(Transformer<? super T, ? extends R> mapper) {
88             try {
89                 return success(mapper.transform(value));
90             } catch (Throwable e) {
91                 return failure(e);
92             }
93         }
94
95         public <R> Try<R> flatMap(Transformer<? super T, ? extends Try<? extends
96             @SuppressWarnings("unchecked")
97             Try<R> t = (Try<R>) mapper.transform(value);
98             return t;
99         }
100
101         public T get() throws Throwable {
102             return value;
103         }
104
105         public Try<T> onFailure(Consumer<? super Throwable> consumer) {
106             return this;
107         }
108
109         public Try<T> recover(Transformer<? super Throwable, ? extends T> transfo
110             return this;
111         }
112
113         public boolean equals(Object o) {
114             if (o != null && o instanceof Success) {
115                 Success<?> success = (Success<?>) o;
116                 if (success.value != null) {
```

```
117         return success.value.equals(this.value);
118     }
119     return this.value == null;
120 }
121 return false;
122 }
123 }
124
125 private static class Failure<T> extends Try<T> {
126     Throwable throwable;
127
128     Failure(Throwable value) {
129         this.throwable = value;
130     }
131
132     public String toString() {
133         return "Failure: " + String.valueOf(throwable);
134     }
135
136     public <R> Failure<R> map(Transformer<? super T, ? extends R> mapper) {
137         @SuppressWarnings("unchecked")
138         Failure<R> t = (Failure<R>) this;
139         return t;
140     }
141
142     public <R> Failure<R> flatMap(Transformer<? super T, ? extends Try<? exte
143         @SuppressWarnings("unchecked")
144         Failure<R> t = (Failure<R>) this;
145         return t;
146     }
147
148     public T get() throws Throwable {
149         throw throwable;
150     }
151
152     public Try<T> onFailure(Consumer<? super Throwable> consumer) {
153         try {
154             consumer.consume(this.throwable);
155             return this;
156         } catch (Throwable t) {
157             return failure(t);
158         }
159     }
160
161     public Try<T> recover(Transformer<? super Throwable, ? extends T> transfo
```

```
162         try {
163             return success(transformer.transform(this.throwable));
164         } catch (Throwable t) {
165             return failure(t);
166         }
167     }
168
169     public boolean equals(Object o) {
170         if (o != null && o instanceof Failure) {
171             Failure<?> that = (Failure<?>) o;
172             return that.throwable.toString().equals(this.throwable.toString());
173         }
174         return false;
175     }
176 }
177 }
```