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

```java
package cs2030s.fp;

import java.util.NoSuchElementException;
import java.util.function.Consumer;

/**
 * CS2030S Lab 5.
 * AY21/22 Semester 2
 *
 * @author Brandon (Group 12A)
 */
public abstract class Maybe<T> {
  private Maybe() {
  }

  public static <T> Maybe<T> of(T t) {
    if (t == null) {
      return none();
    } else {
      return some(t);
    }
  }

  public static <T> Maybe<T> none() {
    @SuppressWarnings("unchecked")
    Maybe<T> none = (Maybe<T>) None.NONE;
    return none;
  }

  public static <T> Maybe<T> some(T t) {
    return new Some<T>(t);
  }

  protected abstract T get() throws NoSuchElementException;

  public abstract Maybe<T> filter(BooleanCondition<? super T> cond);

  public abstract <U> Maybe<U> map(Transformer<? super T, ? extends U> transf

  public abstract <U> Maybe<U> flatMap(Transformer<? super T,
        ? extends Maybe<? extends U>> transformer);

  public abstract T orElse(T t);

  public abstract T orElseGet(Producer<? extends T> producer);

  /**
   * If the value within this Maybe is missing, do nothing.
   * Otherwise, consume the value with the given consumer.
   *
   * @param consumer The consumer to consume the value
   */
  public abstract void consumeWith(Consumer<? super T> consumer);

  private static class None extends Maybe<Object> {

    private None() {
      super();
    }

    private static final Maybe<?> NONE = new None();

    @Override
    protected Object get() throws NoSuchElementException {
      throw new NoSuchElementException();
    }

    @Override
    public Maybe<Object> filter(BooleanCondition<? super Object> cond) {
      return Maybe.none();
    }

    @Override
    public <U> Maybe<U> map(Transformer<? super Object, ? extends U> transfor
      return Maybe.none();
    }

    @Override
    public <U> Maybe<U> flatMap(Transformer<? super Object,
          ? extends Maybe<? extends U>> transformer) {
      return Maybe.none();
    }

    @Override
    public Object orElse(Object o) {
      return o;
    }

    @Override
    public Object orElseGet(Producer<? extends Object> producer) {
      return producer.produce();
    }

    @Override
    public void consumeWith(Consumer<? super Object> consumer) {
    }

    @Override
    public String toString() {
      return "[]";
    }

    @Override
    public boolean equals(Object o) {
      if (this == o) {
        return true;
      }

      if (o == null || !(o instanceof None)) {
        return false;
      }

      return true;
    }
  }

  private static final class Some<T> extends Maybe<T> {
    private T t;

    protected Some(T t) {
      super();
      this.t = t;
    }

    @Override
    protected T get() throws NoSuchElementException {
      return this.t;
    }

    @Override
    public Maybe<T> filter(BooleanCondition<? super T> cond) {
      if (this.t != null && !cond.test(this.t)) {
        return Maybe.none();
      } else {
        return this;
      }
    }

    @Override
    public <U> Maybe<U> map(Transformer<? super T, ? extends U> transformer)
      return Maybe.some(transformer.transform(this.t));
    }

    @Override
    public <U> Maybe<U> flatMap(Transformer<? super T, ? extends Maybe<? exte
      @SuppressWarnings("unchecked")
      Maybe<U> maybeU = (Maybe<U>) transformer.transform(this.t);
      return maybeU;
    }

    @Override
    public T orElse(T t) {
      return this.t;
    }

    @Override
    public T orElseGet(Producer<? extends T> producer) {
      return this.t;
    }

    @Override
    public void consumeWith(Consumer<? super T> consumer) {
      consumer.accept(this.t);
    }

    @Override
    public String toString() {
      return String.format("[%s]", this.t);
    }

    @Override
    public boolean equals(Object o) {
      if (this == o) {
        return true;
      }

      if (o == null || !(o instanceof Some<?>)) {
        return false;
      }

      Some<?> other = (Some<?>) o;

      if (this.t == null && other.t == null) {
        return true;
      } else if (this.t == null) {
        return false;
      } else {
        return this.t.equals(other.t);
      }
    }
  }
}
```