Week 9

16 March 2023

Highlight of `Maybe<T>`

Getting the Type Correct in `Maybe<T>`

```
private static final Maybe<?> NONE = new None();
public static <T> Maybe<T> none() { .. }
public static <T> Maybe<T> some(T t) {.. }
public static <T> Maybe<T> of(T t) { .. }
protected abstract T get();
public abstract Maybe<T> filter(BooleanCondition<? super T> cond);
public abstract <U> Maybe<U> map(Transformer<? super T, ? extends U> transformer);
public abstract <U> Maybe<U> flatMap(Transformer<? super T,
        ? extends Maybe<? extends U>> transformer);
public abstract <U extends T> T orElse(U u);
public abstract T orElseGet(Producer<? extends T> producer);
public abstract void ifPresent(Consumer<? super T> consumer);
```

Some<T>

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

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

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

    @Override
    protected T get() {
      return t;
    }
```

Some<T>

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

@Override
public <U> Maybe<U> flatMap(Transformer<? super T,
        ? extends Maybe<? extends U>> transformer) {
  @SuppressWarnings("unchecked")
  Maybe<U> m = (Maybe<U>) transformer.transform(this.get());
  return m;
}
```

Return Type of `transform`

```
Transformer<? super T,? extends Maybe<? extends U>> t;
t.transform(this.get());
```

`t.transform(this.get())` return some unknown type that is a subtype of `Maybe<? extends U>`, which we cannot assign to `Maybe<U>` without casting.

Alternative to `flatMap` (that does not work)

```
@Override
public <U> Maybe<U> flatMap(Transformer<? super T,
        ? extends Maybe<? extends U>> transformer) {
  return Maybe.of(transformer.transform(this.get()).orElse(null));
}
```

What if `transformer.transform(this.get())` returns `Some(null)`?

```
@Override
public Maybe<T> filter(BooleanCondition<? super T> cond) {
  if (this.get() == null || cond.test(this.get())) {
    return this;
  }
  return Maybe.<T>none();
}

@Override
public <U extends T> T orElse(U u) {
  return this.get();
}

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

@Override
public void ifPresent(Consumer<? super T> consumer) {
  consumer.consume(this.get());
}
```

What you need for Lab 6

- Writing Java docs

Documenting Your Code

- Why: how implementer communicates with clients
- How: specially-formatted comments in the code
- How: use tools to generate HTML documentation
  - Doxygen, Sphinx, etc.
  - We use `javadoc` for Java

A Sample Javadoc Block

```
/**
 * Encapsulate a circle on a 2D plane.  The `Circle` class
 * supports (i) checking if a point is contained in the
 * circle, and (ii) moving the circle to a new position.
 */
```

Javadoc Tags for Documenting a Method

- `@param` for method parameter and type parameter
- `@return` for return value (omit if void)
- `@throws` for checked exception thrown

(Use in the order above)

```java
/**
 * Create an instance of Maybe with a given value t.
 *
 * @param <T> The type of the value in the Some instance.
 * @param t The value to be wrapped within this Maybe container
 * @return A new Maybe instance initialized with value t.
 */
public static <T> Maybe<T> some(T t) {
  return new Some<T>(t);
}
```

```java
/**
 * Return the value within this Maybe if exists; throw an exception
 * otherwise.
 *
 * @return A value in the container
 * @throws NoSuchElementException if this `Maybe` is a `None`
 */
protected abstract T get();
```

You do not have to:

- host the generate the HTML files on a server, but this is, FYI, what the output looks like.
- submit the generated documents.
- write javadoc for self-explanatory, simple, obvious, methods. e.g., getX(), unless you want to explain what x means.

Run `javadoc` to check for errors:

```
javadoc -quiet -private -d docs cs2030s/fp/Lazy.java
```

Note:

- `-quiet`: only errors and warnings are shown
- `-private`: include documentation from all fields/methods
- `-d doc`: put the generated HTML in a subdirectory called `doc`

Goals of Lab 6

- Extend `cs2030s.fp` with `Lazy<T>`
- Practice using `Maybe<T>`
- Practice using lambdas and lazy evaluation

Note:

- `Lazy<T>` is an important component for Lab 7
- This lab is adapted from PE2 19/20 Sem 1

Lab 6

- Please accept Lab 6
- Run `~cs2030s/get-lab6` on PE hosts.
- Solve and submit before Tuesday night

`Lazy<T>` from Lecture

```
class Lazy<T> {
  private T value;           // valid iff evaluated is true
  private boolean evaluated;  // is value valid?
  private Producer<T> producer;

  public Lazy(Producer<T> producer) {
    evaluated = false;
    value = null;
    this.producer = producer;
  }

  public T get() {
    if (!evaluated) {  // check if value is there
      value = producer.produce();
      evaluated = true;
    }
    return value;
  }
```

The `value` in `Lazy<T>` may or may not be there.

We already have an abstraction that takes care of such values for us: `Maybe<T>`

`Lazy<T>` in Lab 6

```
class Lazy<T> {
  private Maybe<T> value;                  // value may or may not be there
  private Producer<? extends T> producer;  // wildcard for extra flexibility
  // no other fields can be added
   :
}
```

Using `Maybe<T>` in `Lazy<T>`

- Avoid checking if `value` is there.
- Avoid using `value.get()` (since it could throw an exception)

```
// Bad
if (!value.equals(Maybe.none()) {
  return value.get();
} else {
  return -1;
}

// Good
return value.orElse(-1);
```

```
jshell> s = () -> { System.out.println("world!"); return "hello"; }
jshell> Lazy<String> hello = Lazy.of(s)
jshell> hello
hello ==> ?
jshell> hello.get()
world!
$.. ==> "hello"

jshell> // check that "world!" should not be printed again.
jshell> hello.get()
$.. ==> "hello"
```

```
jshell> Producer<String> s = () -> "123456"
jshell> Lazy<String> lazy = Lazy.of(s)
jshell> lazy.map(str -> str.substring(0, 1))
$.. ==> ?
jshell> lazy
$.. ==> ?
jshell> lazy.map(str -> str.substring(0, 1)).get()
$.. ==> "1"
jshell> lazy.get()
$.. ==> "123456"
```

Use `Lazy<T>` to Build a Lazy List

```
jshell> Transformer<Integer, Integer> incr = x -> {
   ...>   System.out.println("x + 1");
   ...>   return x + 1;
   ...> }
jshell> LazyList<Integer> l = l.generate(1000000, 0, incr);
jshell> l
l ==> [0, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,  ...]
jshell> l.indexOf(4);
x + 1
x + 1
x + 1
x + 1
$9 ==> 4
jshell> l
l ==> [0, 1, 2, 3, 4, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ...]
jshell> l.get(2)
$13 ==> 2
```

Version: v1.0

Last Updated: Mon Mar 14 23:28:31 +08 2022