Prova 1 de PAOO

Mario Leston

18 de outubro de 2019

Convenção Escrevemos `obj:T` para denotar que o objeto `obj` tem tipo `T`.

Questão 1. [2 pontos] Encontrar os fatores primos de um inteiro $n \geqslant 1$ é uma tarefa cara computacionalmente. Suponha que você precise lidar com números inteiros "grandes" e com suas respectivas fatorações. Assim, é claro, se já foram calculados os fatores primos de um determinado inteiro n, não é necessário repetir tal computação, uma vez que, como já enfatizado, é computacionalmente caro obter tais fatores. Escreva uma classe que implemente esta ideia. É evidente que sua classe deve conter um método que dado um inteiro $n \geqslant 1$ devolva o conjunto dos fatores primos de n. Para ajudá-lo em tal tarefa, eis uma função simplória que obtém os fatores primos de um `n:BigInteger` maior ou igual a 1:

```
Collection<BigInteger> primeFactors(BigInteger n) {
    Collection<BigInteger> factors = new HashSet<>();
    BigInteger d = BigInteger.TWO;
    while (!n.equals(BigInteger.ONE)) {
        if (n.mod(d) == BigInteger.ZERO) {
            factors.add(d);
            n = n.divide(d);
        } else {
            d = d.add(BigInteger.ONE);
        }
    }
    return Collections.unmodifiableCollection(factors);
}
```

Questão 2. [2 pontos] Considere a classe `Maybe` vista em aula:

```
abstract class Maybe<T> {
    private static Maybe empty = nothing();

    public abstract boolean isEmpty();
```

```
    public boolean isPresent() { return !isEmpty(); }
    public abstract T get();
    public abstract T orElseGet(Supplier<T> d);
    public abstract <U> Maybe<U> map(Function<T, U> f);
    public abstract <U> Maybe<U> flatMap(Function<T, Maybe<U>> f);
    public abstract <U> Maybe<U> app(Maybe<Function<T, U>> mfn);

    public static <E> Maybe<E> empty() { return (Maybe<E>) empty; }

    private static <E> Maybe<E> nothing() {
        return new Maybe<E>() {
            public boolean isEmpty() { return true; }
            public E get() { throw new IllegalCallerException(); }
            public E orElseGet(Supplier<E> d) { return d.get(); }
            public <U> Maybe<U> map(Function<E, U> f) { return empty(); }
            public <U> Maybe<U> flatMap(Function<E, Maybe<U>> f) { return empty(); }
            public <U> Maybe<U> app(Maybe<Function<E, U>> mfn) { ??? }
        };
    }

    public static <E> Maybe<E> some(E value) {
        if (value == null) throw new IllegalCallerException();
        return new Maybe<E>() {
            public boolean isEmpty() { return false; }
            public E get() { return value; }
            public E orElseGet(Supplier<E> d) { return value; }
            public <U> Maybe<U> map(Function<E, U> f) { return some(f.apply(value)); }
            public <U> Maybe<U> flatMap(Function<E, Maybe<U>> f) { return f.apply(get()); }
            public <U> Maybe<U> app(Maybe<Function<E, U>> mfn) { ??? }
        };
    }
}
```

Implemente o método `app` que recebe `mfn:Maybe<Function<E, U>>` e devolve o resultado da aplicação de f a x, caso o opcional `mfn` contenha uma função f e o opcional do objeto seja habitado por x; em caso contrário, o método deve evidentemente devolver um opcional vazio. Note que a classe `Optional` do `Java` carece deste importante método.

Questão 3. [6 pontos] Um *buffer com tempo de vida* é uma sequência cujos objetos são dotados de um método *lifetime* que devolve um inteiro positivo, chamado de *tempo de vida* do objeto. Em um buffer podemos realizar duas operações:

(i) inserir um objeto, e
(ii) remover objetos do buffer.

Suponha que um objeto foi inserido em um buffer b. Este objeto só poderá ser removido de b se o seu tempo de permanência em b for maior ou igual ao seu tempo de vida. O tempo de permanência de um objeto em um buffer é zero quando o objeto é inserido no buffer e é incrementado de uma unidade cada vez que a operação (i) ou (ii) é realizada. Assim, a operação de remoção retira do buffer todos os objetos cujo tempo de permanência no buffer é maior ou igual ao seu tempo de vida.

Um buffer *enfeitado* é um buffer com tempo de vida que é dotado de uma operação que devolve os objetos que estão a mais tempo no buffer.

Finalmente, um buffer *limitado* é um buffer enfeitado que admite no buffer um número máximo e fixo de elementos no buffer, de tal forma que quando este limite for atingido não é possível inserir mais elementos no buffer; para que isso seja possível elementos deverão ser primeiro retirados do buffer.

Implemente as interfaces e classes acima especificadas e justifique a estratégia de implementação que você utilizou. Lembre-se que você deve organizar as suas classes de tal forma a maximizar o seu escopo de utilização.