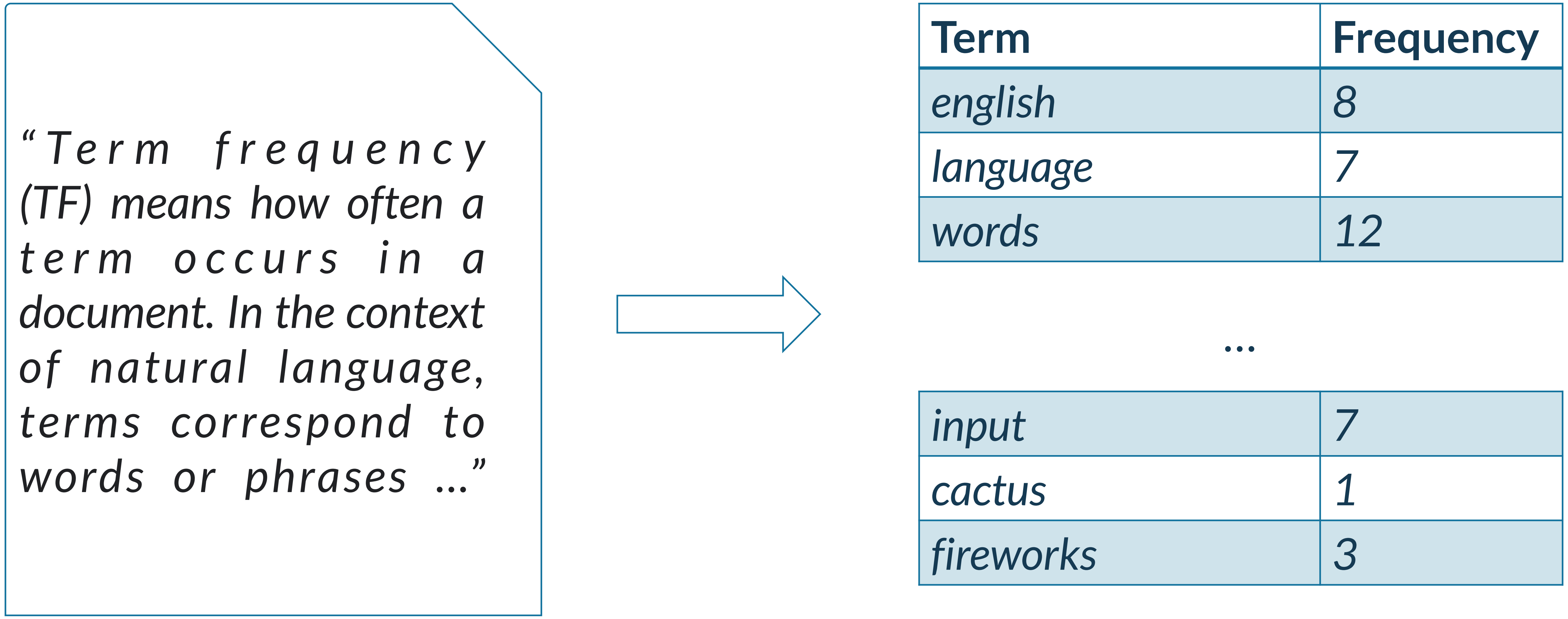
Programming in Java – Solution of assignments 3

Paolo Vercesi

Technical Program Manager

Assignment

Write a class (or a set of classes) that given a string it produces a Term Frequency table.
Consider the option to provide a list of stop words, normalization, etc.
Provide an option to print the table in alphabetical order and by frequency.



"Term frequency (TF) means how often a term occurs in a document. In the context of natural language, terms correspond to words or phrases …"

Term	Frequency
english	*8*
language	*7*
words	*12*

…

input	*7*
cactus	*1*
fireworks	*3*

The TermFrequency class

```java
public class TF {

    private final Map<String, Integer> map;

    public TF(Map<String, Integer> map) {
        this.map = new TreeMap<>(map);
    }

    private void print(Map<String, Integer> map) {
        for (var entry : map.entrySet()) {
            System.out.println(entry);
        }
    }

    …
```

```java
    …

    public void printAlphabetically() {
        print(map);
    }

    public void printByFrequency() {
        Comparator<String> c = new Comparator<>() {
            @Override
            public int compare(String o1, String o2) {
                int f1 = map.get(o1);
                int f2 = map.get(o2);
                return f1 == f2 ? o1.compareTo(o2) : f1-f2;
            }
        };
        Map<String, Integer> treeMap = new TreeMap<>(c);
        treeMap.putAll(map);
        print(treeMap);
    }
}
```

A TermFrequency builder

Write a class (or a set of classes) that given a string it produces a Term Frequency table

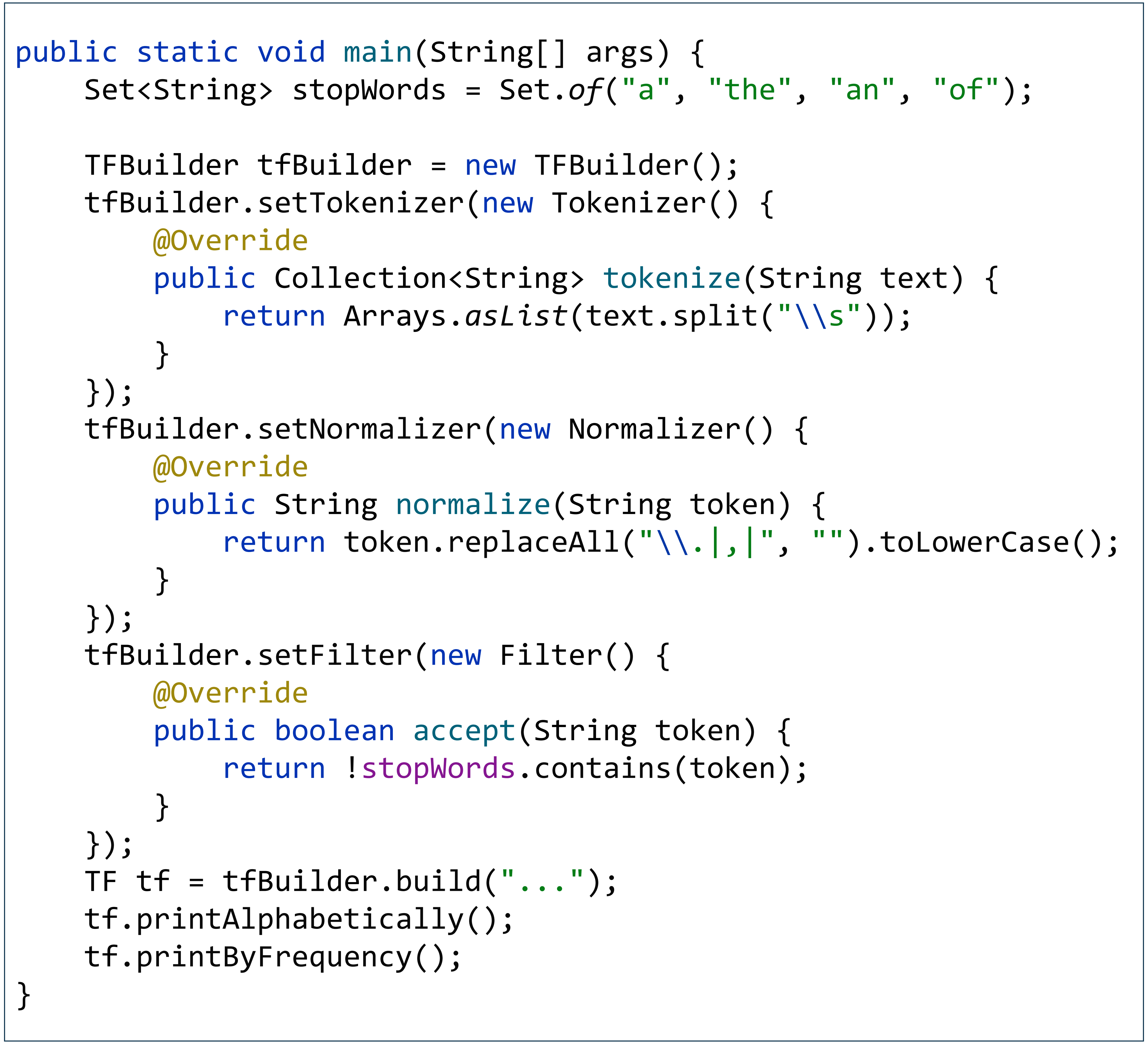
```java
public class TFBuilder {

  private Tokenizer tokenizer;
  private Normalizer normalizer;
  private Filter filter;

  public void setTokenizer(Tokenizer tokenizer) {
     this.tokenizer = tokenizer;
  }

  public void setNormalizer(Normalizer normalizer) {
     this.normalizer = normalizer;
  }

  public void setFilter(Filter filter) {
    this.filter = filter;
  }
  …
```

```java
  …
  TF build(String text) {
    Collection<String> tokens = tokenizer.tokenize(text);
    Map<String, Integer> map = new HashMap<>();
    for (String term : tokens) {
      term = normalizer.normalize(term);
      if (filter.accept(term)) {
        map.merge(term, 1, new BiFunction<>() {
          @Override
          public Integer apply(Integer v, Integer d) {
            return v + 1;
          }
        });
      }
    }
    return new TF(map);
  }
}
```

Tokenizer & C.

```java
public interface Tokenizer {

    Collection<String> tokenize(String text);
}

public interface Normalizer {

    String normalize(String token);
}

public interface Filter {

    boolean accept(String token);
}
```

A usage example



```java
public static void main(String[] args) {
    Set<String> stopWords = Set.of("a", "the", "an", "of");

    TFBuilder tfBuilder = new TFBuilder();
    tfBuilder.setTokenizer(new Tokenizer() {
        @Override
        public Collection<String> tokenize(String text) {
            return Arrays.asList(text.split("\\s"));
        }
    });
    tfBuilder.setNormalizer(new Normalizer() {
        @Override
        public String normalize(String token) {
            return token.replaceAll("\\.|,|", "").toLowerCase();
        }
    });
    tfBuilder.setFilter(new Filter() {
        @Override
        public boolean accept(String token) {
            return !stopWords.contains(token);
        }
    });
    TF tf = tfBuilder.build("...");
    tf.printAlphabetically();
    tf.printByFrequency();
}
```

Thank you!

esteco.com