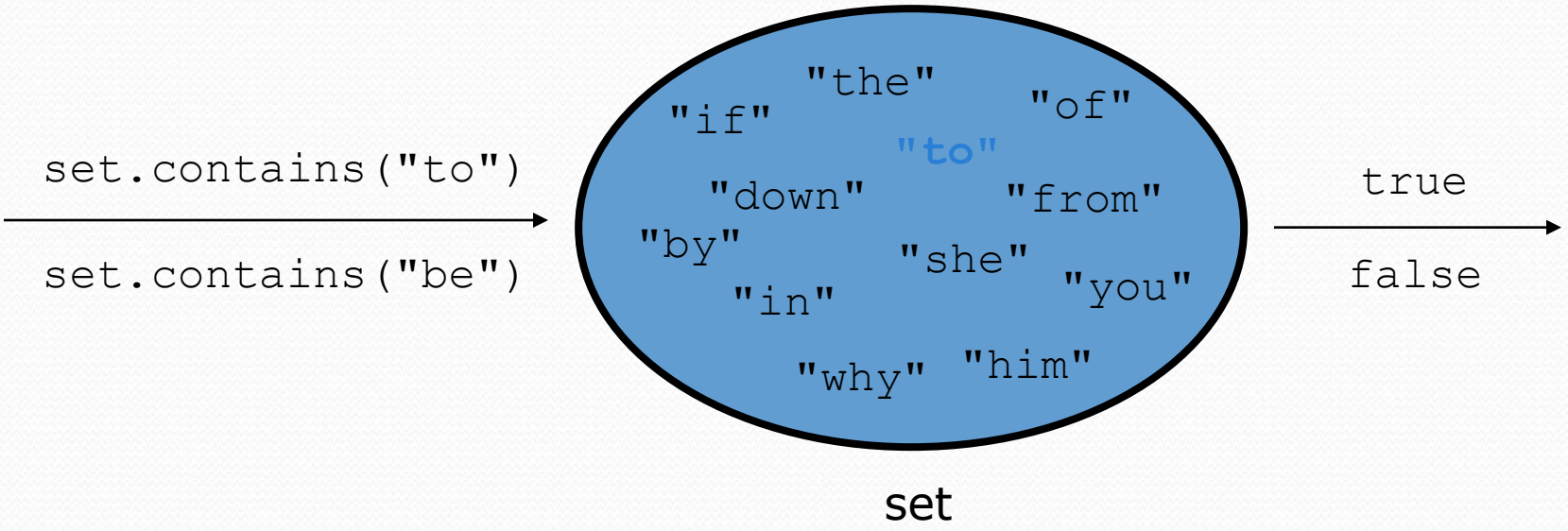
# Exercise

- Write a program that counts the number of unique words in a large text file (say, *Moby Dick* or the King James Bible).
  - Store the words in a collection and report the # of unique words.
  - Once you've created this collection, allow the user to search it to see whether various words appear in the text file.

- What collection is appropriate for this problem?

# Sets (11.2)

- **set**: A collection of unique values (no duplicates allowed) that can perform the following operations efficiently:
  - add, remove, search (contains)
  - We don't think of a set as having indexes; we just add things to the set in general and don't worry about order



# `Set` methods

In Java, Set is an interface that allows you to call the following methods

| | |
|---|---|
| `add(`**value**`)` | adds the given value to the set |
| `contains(`**value**`)` | returns `true` if the given value is found in this set |
| `remove(`**value**`)` | removes the given value from the set |
| `clear()` | removes all elements of the set |
| `size()` | returns the number of elements in list |
| `isEmpty()` | returns `true` if the set's size is 0 |
| `toString()` | returns a string such as `"[3, 42, -7, 15]"` |

# `Set` implementation

- in Java, sets are represented by `Set` type in `java.util`
- `Set` is implemented by `HashSet` and `TreeSet` classes
  - `HashSet`: implemented using a "hash table" array;
    very fast: **O(1)** for all operations
    elements are stored in unpredictable order
  - `TreeSet`: implemented using a "binary search tree";
    pretty fast: **O(log N)** for all operations
    elements are stored in sorted order

```
Set<Integer> numbers = new TreeSet<Integer>();
Set<String> words = new HashSet<String>();
```

# The "for each" loop (7.1)

```
for (type name : collection) {
    statements;
}
```

- Provides a clean syntax for looping over the elements of a `Set`, `List`, array, or other collection

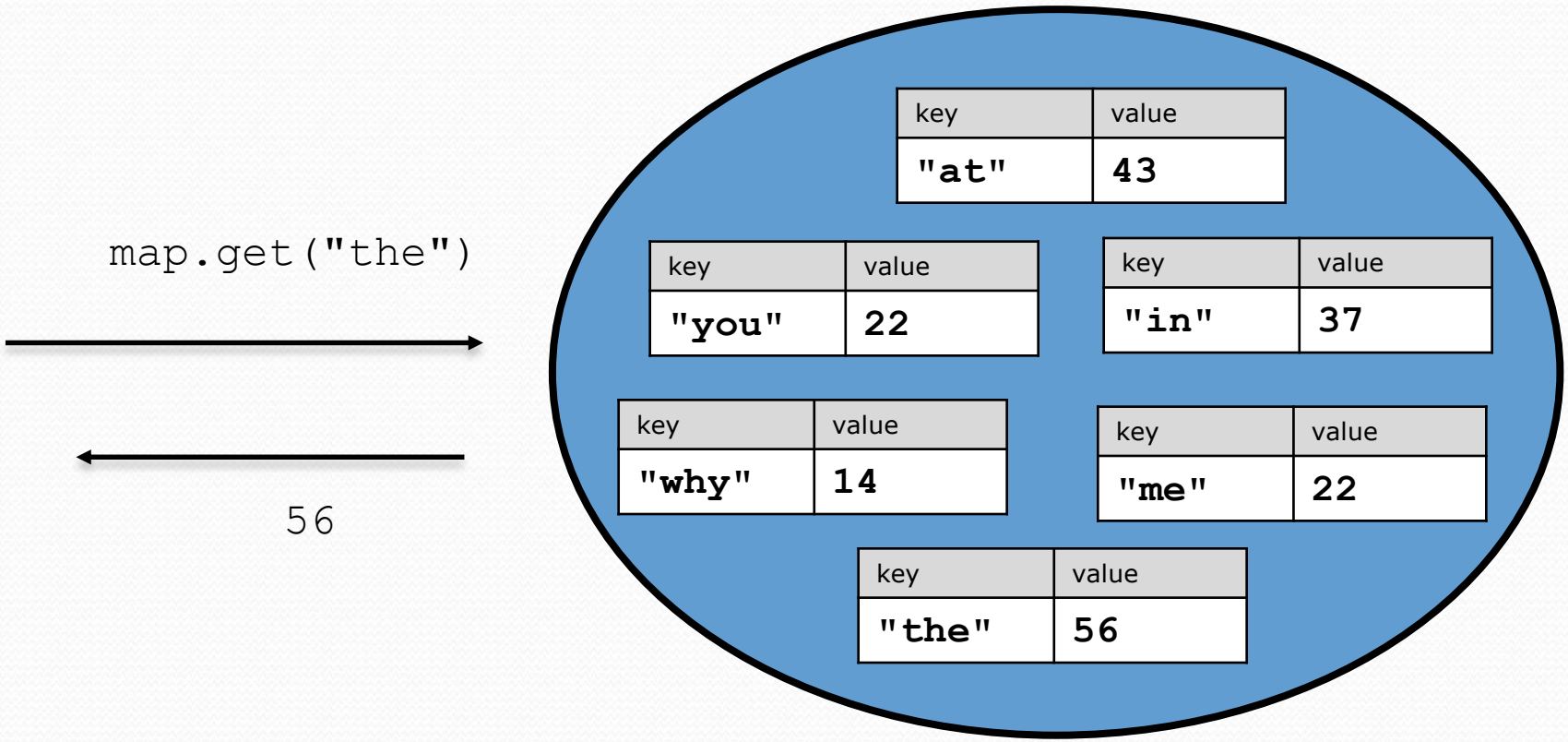
```
Set<Double> grades = new HashSet<Double>();
...

for (double grade : grades) {
    System.out.println("Student's grade: " + grade);
}
```

  - needed because sets have no indexes; can't `get` element `i`

# Maps (11.3)

- **map**: Holds a set of key-value pairs, where each key is unique
  a.k.a. "dictionary", "associative array", "hash"



# `Map` implementation

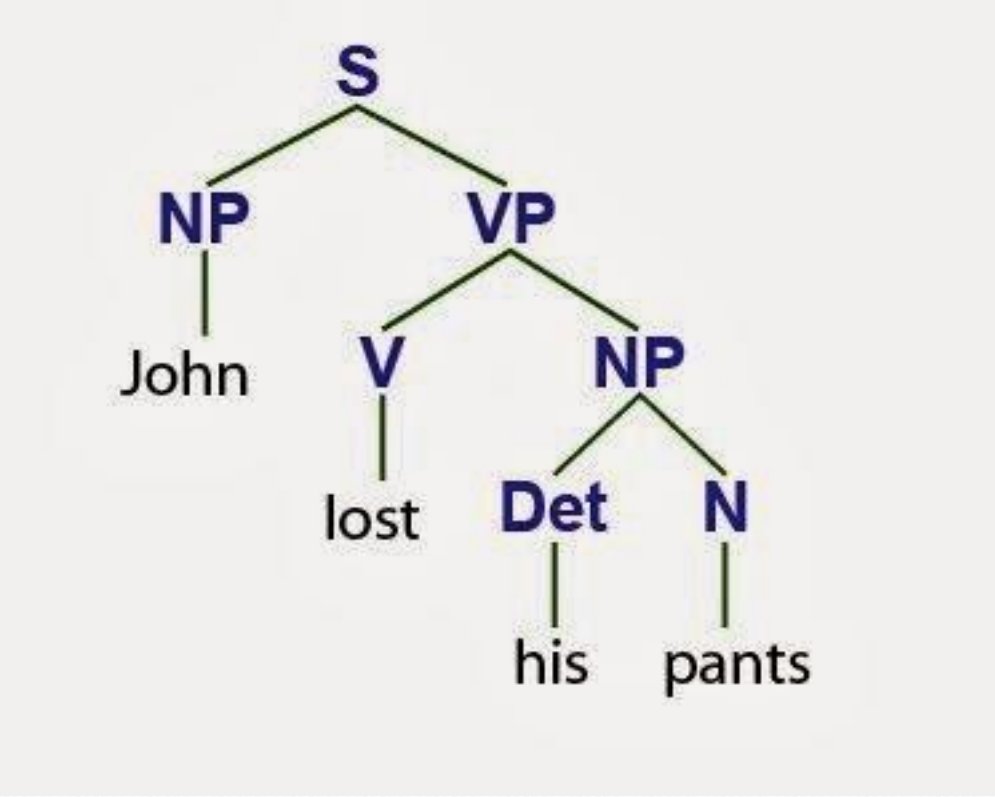
- in Java, maps are represented by `Map` type in `java.util`
- `Map` is implemented by the `HashMap` and `TreeMap` classes
  - `HashMap`: implemented using an array called a "hash table"; extremely fast: **O(1)** ; keys are stored in unpredictable order
  - `TreeMap`: implemented as a linked "binary tree" structure; very fast: **O(log N)** ; keys are stored in sorted order
  - `LinkedHashMap`: O(1) ; keys are stored in order of insertion
- Maps require 2 type params: one for keys, one for values.

```
// maps from String keys to Integer values
Map<String, Integer> votes = new HashMap<String, Integer>();
// maps from Integer keys to String values
Map<Integer, String> words = new TreeMap<Integer, String>();
```

# `Map` methods

| | |
|---|---|
| `put(`**key**`, `**value**`)` | adds a mapping from the given key to the given value;<br>if the key already exists, replaces its value with the given one |
| `get(`**key**`)` | returns the value mapped to the given key (`null` if not found) |
| `containsKey(`**key**`)` | returns `true` if the map contains a mapping for the given key |
| `remove(`**key**`)` | removes any existing mapping for the given key |
| `clear()` | removes all key/value pairs from the map |
| `size()` | returns the number of key/value pairs in the map |
| `isEmpty()` | returns `true` if the map's size is 0 |
| `toString()` | returns a string such as `"{a=90, d=60, c=70}"` |

| | |
|---|---|
| `keySet()` | returns a set of all keys in the map |
| `values()` | returns a collection of all values in the map |
| `putAll(`**map**`)` | adds all key/value pairs from the given map to this map |
| `equals(`**map**`)` | returns `true` if given map has the same mappings as this one |


S
NP
VP
John
V
NP
lost
Det
N
his
pants

# Languages and grammars

- (formal) **language**: A set of words or symbols.

- **grammar**: A description of a language that describes which sequences of symbols are allowed in that language.
  - describes language *syntax* (rules) but not *semantics* (meaning)
  - can be used to generate strings from a language, or to determine whether a given string belongs to a given language

# Backus-Naur (BNF)

- **Backus-Naur Form (BNF)**: A syntax for describing language grammars in terms of transformation *rules*, of the form:

  <**symbol**> ::= <**expression**> | <**expression**> *...* | <**expression**>

  - **terminal**: A fundamental symbol of the language.
  - **non-terminal**: A high-level symbol describing language syntax, which can be transformed into other non-terminal or terminal symbol(s) based on the rules of the grammar.

  - developed by two Turing-award-winning computer scientists in 1960 to describe their new ALGOL programming language

# Sentence generation