Review: Maps

- **map**: Holds a set of unique keys and a collection of values, where each key is associated with one value.
  - a.k.a. "dictionary", "associative array", "hash"

- basic map operations:
  - `put(key, value)`: Adds a mapping from a key to a value.
  - `get(key)`: Retrieves the value mapped to the key.
  - `remove(key)`: Removes the given key and its reference to the mapped value.
  - `containsKey(key)`: Returns true if key is present.
  - `keySet()`: Returns a Set of all keys.

- Create a Map:

```java
Map<String, Integer> counts = new HashMap<String, Integer>();
```

```java
Map<String, Integer> counts = new TreeMap<String, Integer>();
```

Exercise solution

```java
// read file into a map of [word --> number of occurrences]
Map<String, Integer> wordCount = new HashMap<String, Integer>();
Scanner input = new Scanner(new File("hamlet.txt"));
while (input.hasNext()) {
    String word = input.next();
    if (wordCount.containsKey(word)) {
        // seen this word before; increase count by 1
        int count = wordCount.get(word);
        wordCount.put(word, count + 1);
    } else {
        // never seen this word before
        wordCount.put(word, 1);
    }
}
Scanner console = new Scanner(System.in);
System.out.print("Word to search for? ");
String word = console.next();
System.out.println("appears " + wordCount.get(word) + " times.");
```

Python and Perl

Maps/Dictionaries are a built-in feature of many scripting languages!

**Python:**

```python
wordCount = {}
...
if wordCount.has_key(word):
    wordCount[word] += 1
else:
    wordCount[word] = 1
```

**Perl:**

```perl
my %wordcount = ();
...
if (defined($wordCount{$word})) {
    $wordCount{$word}++;  
} else {
    $wordCount{$word} = 1;
}
```

Example: The Oscars!

Winners of the 84th Annual Academy Awards:
- Best Picture: The Artist
- Best Actor: Jean Dujardin, The Artist
- Best Actress: Meryl Streep, The Iron Lady
- Best Visual Effects: Hugo
- Best Art Direction: Hugo
- Best Cinematography: Hugo

```java
Map<String, String> oscars = new TreeMap<String, String>();
oscars.put("Cinematography", "Hugo");
oscars.put("Best Visual Effects", "Hugo");
oscars.put("Best Picture", "The Artist");
```
Problem: opposite mapping

- Suppose we want to ask which movies got which awards.
  - How would we structure a map for that?

```java
Map<String, String> oscarsWon = new HashMap<String, String>();
oscarsWon.put("Hugo", "Best Cinematography");
oscarsWon.put("Hugo", "Best Visual Effects");
oscarsWon.put("The Artist", "Best Picture");
...
System.out.println("Hugo won: " + oscarsWon.get("Hugo")); // ???
```

- Problem: A movie, such as "Hugo," can win multiple Oscars.

Proper Map Reversal

- Map each movie to a collection of awards:

```java
Map<String, Set<String>> oscarsWon = new HashMap<String, Set<String>>();
oscarsWon.put("Hugo", new HashSet<String>());
oscarsWon.get("Hugo").add("Best Cinematography");
oscarsWon.get("Hugo").add("Best Visual Effects");
oscarsWon.put("The Artist", new HashSet<String>());
oscarsWon.get("The Artist").add("Best Picture");
...
System.out.println("Hugo won: " + oscarsWon.get("Hugo"));
// [Best Cinematography, Best Visual Effects]
```

(Must be careful to initialize the set for a given movie before adding.)

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:

  \[
  \langle \text{symbol} \rangle ::= \langle \text{expression} \rangle \mid \langle \text{expression} \rangle \ldots \mid \langle \text{expression} \rangle
  \]

  - **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.

- Java grammar

### An example BNF grammar

- **BNF grammar version 2**

  \[
  \langle \text{s} \rangle ::= \langle \text{np} \rangle \langle \text{v} \rangle \\
  \langle \text{np} \rangle ::= \langle \text{pn} \rangle \mid \langle \text{dp} \rangle \langle \text{n} \rangle \\
  \langle \text{pn} \rangle ::= \text{Marty} \mid \text{Stuart} \mid \text{Victoria} \mid \text{Watson} \\
  \langle \text{dp} \rangle ::= \text{a} \mid \text{the} \\
  \langle \text{n} \rangle ::= \text{ball} \mid \text{hamster} \mid \text{carrot} \mid \text{computer} \\
  \langle \text{v} \rangle ::= \text{cried} \mid \text{slept} \mid \text{won Jeopardy}
  \]

  - Some sentences that could be generated from this grammar:
    - the carrot cried
    - Watson won Jeopardy
    - a computer slept

- **BNF grammar version 3**

  \[
  \langle \text{s} \rangle ::= \langle \text{np} \rangle \langle \text{v} \rangle \\
  \langle \text{np} \rangle ::= \langle \text{pn} \rangle \mid \langle \text{dp} \rangle \langle \text{adj} \rangle \langle \text{n} \rangle \\
  \langle \text{pn} \rangle ::= \text{Marty} \mid \text{Stuart} \mid \text{Victoria} \mid \text{Watson} \\
  \langle \text{dp} \rangle ::= \text{a} \mid \text{the} \\
  \langle \text{adj} \rangle ::= \text{silly} \mid \text{invisible} \mid \text{loud} \mid \text{romantic} \\
  \langle \text{n} \rangle ::= \text{ball} \mid \text{hamster} \mid \text{carrot} \mid \text{computer} \\
  \langle \text{v} \rangle ::= \text{cried} \mid \text{slept} \mid \text{won Jeopardy}
  \]

  - Some sentences that could be generated from this grammar:
    - the invisible carrot cried
    - Watson won Jeopardy
    - a computer slept
    - a romantic ball won Jeopardy
Grammars and recursion

- Grammar rules can be defined recursively, so that the expansion of a symbol can contain that same symbol.
  - There must also be expressions that expand the symbol into something non-recursive, so that the recursion eventually ends.

Grammar, final version

- Could this grammar generate the following sentences?
  Fred honored the green wonderful child
  big Jane wept the fat man fat
- Generate a random sentence using this grammar.
Iterators (11.1)

- **iterator**: An object that allows a client to traverse the elements of a collection, regardless of its implementation.
  - Remembers a position within a collection, and allows you to:
    - get the element at that position
    - advance to the next position
    - (possibly) remove or change the element at that position
  - Benefit: A common way to examine any collection's elements.

![Index and data diagram]

### Iterator methods

- hasNext(): returns `true` if there are more elements to examine
- next(): returns the next element from the collection (throws a NoSuchElementException if there are none left to examine)
- remove(): removes from the collection the last value returned by next() (throws IllegalStateException if you have not called next() yet)

Every provided collection has an iterator method:

```java
Set<String> set = new HashSet<String>();
...
Iterator<String> itr = set.iterator();
...
```

Iterators: Example

```
List<String> list = new LinkedList<String>();

// Method 1: get(i)
for (int i = 0; i < list.size(); i++) {
    String s = list.get(i);
    System.out.println(s);
}

// Method 2: for-each loop
for (String s : list) {
    System.out.println(s);
}

// Method 3: iterators
Iterator<String> iter = list.iterator();
while (iter.hasNext()) {
    String s = iter.next();
    System.out.println(s);
}
```

```
List<String> list1 = new LinkedList<String>();
List<String> list2 = new LinkedList<String>();

// Method 1: get(i)
for (int i = 0; i < list1.size(); i++) {
    String s1 = list1.get(i);
    String s2 = list2.get(i);
    System.out.println(s1 + " : " + s2);
}

// Method 2: for-each loop
for (String s : list1) {
    System.out.println(s);
}

// Method 3: iterators
Iterator<String> iter1 = list1.iterator();
Iterator<String> iter2 = list2.iterator();
while (iter1.hasNext()) {
    String s1 = iter1.next();
    String s2 = iter2.next();
    System.out.println(s1 + " : " + s2);
}
```
for-each loop and Iterable

- Java's collections can be iterated using a "for-each" loop:

  ```java
  List<String> list = new LinkedList<String>();
  ...
  for (String s : list) {
    System.out.println(s); // Current position in list
  }
  ```

- This works for any class that implements the `Iterable` interface:

  ```java
  public interface Iterable<E> {
    public Iterator<E> iterator();
  }
  ```

  For example, we could modify `ArrayIntList` and `LinkedIntList` to allow iterators and for-each loops.

Array list iterator

```java
public class ArrayList<E> extends AbstractList<E> {
  ...
  // not perfect; doesn't forbid multiple removes in a row
  private class ArrayIterator implements Iterator<E> {
    private int index; // Current position in list
    public ArrayIterator() {
      index = 0;
    }
    public boolean hasNext() { return index < size(); }
    public E next() { index++; return get(index - 1); }
    public void remove() { ArrayList.this.remove(index - 1); index--; }
  }
}
```

Linked list iterator

```java
public class LinkedList<E> extends AbstractList<E> {
  ...
  // not perfect; doesn't support remove
  private class LinkedIterator implements Iterator<E> {
    private ListNode current; // Current position in list
    public LinkedIterator() {
      current = front;
    }
    public boolean hasNext() { return current != null; }
    public E next() { E result = current.data; current = current.next; return result; }
    public void remove() { // Not implemented for now
      throw new UnsupportedOperationException();
    }
  }
}
```