Map
The Map component family allows you to manipulate *mappings* from *keys* (of any type $K$) to *values* (of any type $V$)

- A Map variable holds a very simple “database” of keys and their associated values
- Example: If you need to keep track of the exam grade for each student, you might use a Map$<$String,$Integer$$>$ variable
Interfaces and Classes

`Standard` extends `MapKernel`
`Iterable` extends `MapKernel`
`MapKernel` extends `Map`

`Map` implements `Map1L`
`Map` implements `Map2`
`Map` implements `Map3`
Interfaces and Classes

- **Standard**
  - Implements `Map`
  - Extends `Iterable`
  - Extends `Standard`
  - Has contracts for three methods:
    - `clear`
    - `newInstance`
    - `transferFrom`

- **Iterable**
  - Extends `Map`

- **Map**
  - Implements `Iterable`
  - Extends `Standard`

- **Kernel**
  - Extends `Map`

- **Map1L**
  - Implements `Map`

- **Map2**
  - Implements `Map`

- **Map3**
  - Implements `Map`
MapKernel has contracts for six methods:

- add
- remove
- removeAny
- value
- hasKey
- size
Map has contracts for five other methods:
- replaceValue
- key
- hasValue
- sharesKeyWith
- combineWith
Interfaces and Classes

- `Standard`
- `Iterable`
- `MapKernel`
- `Map`
- `Map1L`
- `Map2`
- `Map3`

`Iterable` has a contract for one method: `iterator`
Mathematical Model

• The value of a Map variable is modeled as a finite set of ordered pairs of type \((K, V)\) with “the function property”, i.e., no two pairs in the set have the same \(K\) value.
  
  – This is sometimes called a (finite) **partial function from** \(K\) **to** \(V\)
Partial Function

\texttt{PARTIAL\_FUNCTION \textit{is finite set of}}
\begin{align*}
\text{(key: } K, \text{ value: } V) \notag \\
\text{exemplar } m \notag \\
\text{constraint} \notag \\
\text{for all } \text{key}_1, \text{key}_2: K, \text{value}_1, \text{value}_2: V \notag \\
\text{where } ((\text{key}_1, \text{value}_1) \text{ is in } m \text{ and } \notag \\
(\text{key}_2, \text{value}_2) \text{ is in } m) \\(\text{if } \text{key}_1 = \text{key}_2 \text{ then } \text{value}_1 = \text{value}_2) \notag \\
\end{align*}
**Partial Function**

This formally states “the function property” for a set of ordered pairs.

```
PARTIAL_FUNCTION is defined as:
  (key: K, value: V)

exemplar m

constraint

for all key1, key2: K, value1, value2: V
  where ((key1, value1) is in m and
         (key2, value2) is in m)
  (if key1 = key2 then value1 = value2)
```
Domain of a (Partial) Function

DOMAIN ( 
    m: PARTIAL_FUNCTION
): finite set of K

satisfies

    for all key: K

    (key is in DOMAIN(m) iff
     there exists value: V
     ((key, value) is in m))
Range of a (Partial) Function

\[ \text{RANGE} \ ( \]
\[ m: \text{PARTIAL\_FUNCTION} \]
\[ ) : \text{finite set of } V \]
\[ \text{satisfies} \]
\[ \text{for all value: } V \]
\[ (\text{value is in } \text{RANGE}(m) \text{ iff there exists key: } K \]
\[ ((\text{key, value}) \text{ is in } m) ) \]
Mathematical Model

- Formally:

  \texttt{type Map is modeled by PARTIAL\_FUNCTION}
No-argument Constructor

• Ensures:

\[
this = \{ \}
\]
**Example**

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Map&lt;String, Integer&gt; m = new Map1L&lt;&gt;();</code></td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map&lt;String,Integer&gt; m = new Map1L&lt;&gt;();</td>
<td>m = { }</td>
</tr>
</tbody>
</table>
add

```java
void add(K key, V value)
```

- Adds the pair \((key, value)\) to `this`
- Aliases: references `key, value`
- Updates: `this`
- Requires:
  - `key` is not in `DOMAIN(this)`
- Ensures:
  - `this = #this union { (key, value) }`
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| m = {("PB", 99),
      ("BW", 17)}
|               |
| k = "PS"      |
| v = 99        |
| m.add(k, v);  |                              |
Example

Is the requires clause satisfied? What is $\text{DOMAIN}(m)$?

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
</table>
| $m = \{("PB", 99),
          ("BW", 17)\}$ |
| $k = "PS"$     |
| $v = 99$       |

$m$.add($k$, $v$);
### Example

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = {(&quot;PB&quot;, 99),\newline \quad \quad (&quot;BW&quot;, 17)}$</td>
</tr>
<tr>
<td></td>
<td>$k = &quot;PS&quot;$</td>
</tr>
<tr>
<td></td>
<td>$v = 99$</td>
</tr>
<tr>
<td><code>m.add(k, v);</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$m = {(&quot;PB&quot;, 99),\newline \quad \quad (&quot;BW&quot;, 17),\newline \quad \quad (&quot;PS&quot;, 99)}$</td>
</tr>
<tr>
<td></td>
<td>$k = &quot;PS&quot;$</td>
</tr>
<tr>
<td></td>
<td>$v = 99$</td>
</tr>
</tbody>
</table>
Example

Note the aliases created here, which you cannot see in the tracing table; you should be able to draw the appropriate diagram showing them.

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = { (&quot;PB&quot;, 99),\quad (&quot;BW&quot;, 17) }$</td>
</tr>
<tr>
<td>$k = &quot;PS&quot;$</td>
</tr>
<tr>
<td>$v = 99$</td>
</tr>
</tbody>
</table>

\[
m = \{ ("PB", 99),\quad ("BW", 17),\quad ("PS", 99) \} \\
k = "PS" \\
v = 99
\]
Another Interface

• The \texttt{Map} interface includes an interface for another related generic type, \texttt{Map.Pair}

• Its mathematical model is simply an ordered pair of a key and a value

• Formally:

\texttt{type Map.Pair is modeled by}

\texttt{(key: K, value: V)}
Map.Pair Methods

- This (immutable) type has only a constructor (taking a $K$ and a $V$) and a \textit{getter} method for each pair component
  - $K$ \texttt{key()}
    - Returns the first component of \texttt{this}
    - Aliases: reference returned by \texttt{key}
  - $V$ \texttt{value()}
    - Returns the second component of \texttt{this}
    - Aliases: reference returned by \texttt{value}
remove

Map.Pair<K,V> remove(K key)

• Removes from this the pair whose first component is key and returns it.

• Updates: this

• Requires: 
  
  key is in DOMAIN(this)

• Ensures: 
  
  remove.key = key and
  remove is in #this and 
  this = #this \ {remove}
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| m = { ("PB", 99),  
     ("BW", 17)  
}                                   | m = { ("PB", 99), 
     ("BW", 17) 
}                                         |
| k = "BW"                            |                                     |
| Map.Pair<String,Integer> p = m.remove(k); |                                     |
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>State</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>m = { (&quot;PB&quot;, 99), (&quot;BW&quot;, 17) }</td>
<td>m = { (&quot;PB&quot;, 99), (&quot;BW&quot;, 17) }</td>
</tr>
<tr>
<td>k = &quot;BW&quot;</td>
<td>k = &quot;BW&quot;</td>
</tr>
<tr>
<td>Map.Pair&lt;String, Integer&gt; p = m.remove(k);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>m = { (&quot;PB&quot;, 99) }</td>
<td>m = { (&quot;PB&quot;, 99) }</td>
</tr>
<tr>
<td>k = &quot;BW&quot;</td>
<td>k = &quot;BW&quot;</td>
</tr>
<tr>
<td>p = (&quot;BW&quot;, 17)</td>
<td>p = (&quot;BW&quot;, 17)</td>
</tr>
</tbody>
</table>
removeAny

Map.Pair<K,V> removeAny()

- Removes and returns an arbitrary pair from this.
- Updates: this
- Requires:
  \[ |this| > 0 \]
- Ensures:
  
  removeAny is in #this and this = #this \{removeAny\}
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = {(&quot;PB&quot;, 99), \quad } $</td>
</tr>
<tr>
<td></td>
<td>$ (&quot;BW&quot;, 17), \quad } $</td>
</tr>
<tr>
<td></td>
<td>$ (&quot;PS&quot;, 99) } $</td>
</tr>
<tr>
<td>Map.Pair&lt;String, Integer&gt; p = m.removeAny();</td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map.Pair&lt;String,Integer&gt; p = m.removeAny();</td>
<td>m = {(&quot;PB&quot;, 99),</td>
</tr>
<tr>
<td></td>
<td>(&quot;BW&quot;, 17),</td>
</tr>
<tr>
<td></td>
<td>(&quot;PS&quot;, 99)</td>
</tr>
<tr>
<td></td>
<td>p = (&quot;PS&quot;, 99)</td>
</tr>
</tbody>
</table>
value

V value(K key)

- Reports the value associated with key in this.
- Aliases: reference returned by value
- Requires:
  \( \text{key is in } \text{DOMAIN(this)} \)
- Ensures:
  \( (\text{key, value}) \text{ is in this} \)
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m = {(\text{&quot;PB&quot;}, 99),)&lt;br&gt;(\text{&quot;BW&quot;}, 17)})</td>
<td>(m = {(\text{&quot;PB&quot;}, 99),)&lt;br&gt;(\text{&quot;BW&quot;}, 17)})</td>
</tr>
<tr>
<td>(k = \text{&quot;PB&quot;})</td>
<td>(k = \text{&quot;PB&quot;})</td>
</tr>
<tr>
<td>(v = -423)</td>
<td>(v = -423)</td>
</tr>
<tr>
<td>(v = m.value(k);)</td>
<td>(v = m.value(k);)</td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| \[ v = m\.value(k); \] | \[ m = \{("PB", 99),\]  
| | \[ ("BW", 17)\} \]  
| | \[ k = "PB" \]  
| | \[ v = -423 \]  
| | \[ m = \{("PB", 99),\]  
| | \[ ("BW", 17)\} \]  
| | \[ k = "PB" \]  
| | \[ v = 99 \]  

\[ m = \{("PB", 99),\]  
\[ ("BW", 17)\} \]  
\[ k = "PB" \]  
\[ v = -423 \]  
\[ m = \{("PB", 99),\]  
\[ ("BW", 17)\} \]  
\[ k = "PB" \]  
\[ v = 99 \]
Example

<table>
<thead>
<tr>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m = { (&quot;PB&quot;, 99), (&quot;BW&quot;, 17) } )</td>
</tr>
<tr>
<td>( k = &quot;PB&quot; )</td>
</tr>
<tr>
<td>( v = -423 )</td>
</tr>
</tbody>
</table>

Note the alias created here, which you cannot see in the tracing table; you should be able to draw the appropriate diagram showing it.
hasKey

boolean hasKey(K key)

• Reports whether there is a pair in this whose first component is key.

• Ensures:

\[
\text{hasKey} = (\text{key is in DOMAIN(this)})
\]
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>`m = { (&quot;PB&quot;, 99),</td>
<td>`m = { (&quot;PB&quot;, 99),</td>
</tr>
<tr>
<td>(&quot;BW&quot;, 17) }`</td>
<td>(&quot;BW&quot;, 17) }`</td>
</tr>
<tr>
<td></td>
<td><code>k = &quot;PB&quot;</code></td>
</tr>
<tr>
<td>`boolean b =</td>
<td></td>
</tr>
<tr>
<td>m.hasKey(k);`</td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| ```java
boolean b =
mHasKey(k);
``` | ```java
m = {("PB", 99),
     ("BW", 17)}
k = "PB"
```

| | ```java
m = {("PB", 99),
     ("BW", 17)}
k = "PB"
b = true
``` |
```c
int size()

• Reports the size (cardinality) of this.
• Ensures:

  size = |this|
```
replaceValue

V replaceValue(K key, V value)

• Replaces the value associated with key in this by value, and returns the old value.

• Aliases: reference value

• Updates: this

• Requires:

key is in DOMAIN(this)

• Ensures:

this = (#this \ {(key, replaceValue)})

union { (key, value) } and

(key, replaceValue) is in #this
<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| m = {("PB", 99), 
    ("BW", 17)}                                      | m = {("PB", 99), 
    ("BW", 17)}                                      |
| k = "PB"                                                            | k = "PB"                                                            |
| v = 85                                                              | v = 85                                                              |
| Integer oldV = 
    m.replaceValue(k, v);                                | Integer oldV = 
    m.replaceValue(k, v);                                |
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| Integer oldV = m.replaceValue(k, v); | m = {("PB", 85), ("BW", 17)}  
k = "PB"  
v = 85 |

oldV = 99
Example

<table>
<thead>
<tr>
<th>k</th>
<th>v</th>
<th>oldV</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;PB&quot;</td>
<td>85</td>
<td>99</td>
</tr>
<tr>
<td>&quot;BW&quot;</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

Note the alias created here, which you cannot see in the tracing table; you should be able to draw the appropriate diagram showing it.
### Another Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| \( v = m.replaceValue(k, v); \) | \( m = \{ ("PB", 99), ("BW", 17) \} \)
|     | \( k = "PB" \)
|     | \( v = 85 \) |
## Another Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| $v = m.replaceValue(k, v);$ | $m = \{("PB", 99),
            ("BW", 17)\}$ |
|     | $k = "PB"
    | $v = 85$ |
| $v = m.replaceValue(k, v);$ | $m = \{("PB", 85),
            ("BW", 17)\}$ |
|     | $k = "PB"
    | $v = 99$ |
This use of the method avoids creating an alias: it **swaps** \( v \) with the value in \( m \) that was previously associated with \( k \).

\[
v = m.replaceValue(k, v);
\]
K key(V value)

• Reports some key associated with value in this.

• Aliases: reference returned by key

• Requires:
  value is in RANGE(this)

• Ensures:
  (key, value) is in this
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| `m = {("PB", 99),
       ("BW", 17)}` | `m = {("PB", 99),
       ("BW", 17)}` |
| `k = "xyz"`  | `k = "xyz"`       |
| `v = 99`     | `v = 99`          |
| `k = m.key(v);` |                  |
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| k = m.key(v); | m = {("PB", 99),
      ("BW", 17)}
     k = "xyz"
     v = 99 |
|       | m = {("PB", 99),
      ("BW", 17)}
     k = "PB"
     v = 99 |
The method `value` is part of the intended use of a `Map` and is efficient in most classes that implement `Map`; the method `key` is rarely of interest and is inefficient in most classes that implement `Map`. 
hasValue

boolean hasValue(V value)

• Reports whether there is a pair in this whose second component is value.

• Ensures:

\[
\text{hasValue} = \ (\text{value is in RANGE(this)})
\]

25 March 2021
OSU CSE
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>boolean b =</code></td>
<td><code>m = {(&quot;PB&quot;, 99),</code></td>
</tr>
<tr>
<td><code>m.hasValue(v);</code></td>
<td><code>(&quot;BW&quot;, 17)}</code></td>
</tr>
<tr>
<td></td>
<td><code>v = 17</code></td>
</tr>
</tbody>
</table>

```java
m = {("PB", 99),
    ("BW", 17)}
v = 17
```
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = { (&quot;PB&quot;, 99), (&quot;BW&quot;, 17) }$</td>
<td>$m = { (&quot;PB&quot;, 99), (&quot;BW&quot;, 17) }$</td>
</tr>
<tr>
<td>$v = 17$</td>
<td>$v = 17$</td>
</tr>
<tr>
<td><strong>boolean</strong> $b =$</td>
<td></td>
</tr>
<tr>
<td>$m$.hasValue($v$);</td>
<td></td>
</tr>
<tr>
<td>$b =$ <strong>true</strong></td>
<td>$b =$ <strong>true</strong></td>
</tr>
</tbody>
</table>
**Code Example:**

```java
boolean b = m.hasValue(v);
```

The method **hasKey** is part of the *intended* use of a **Map** and is efficient in most classes that implement **Map**; the method **hasValue** is rarely of interest and is inefficient in most classes that implement **Map**.

| m = {("PB", 99),
|      ("BW", 17)}
| v = 17
| b = **true**
void combineWith(Map<K,V> m)

• Combines m with this.
• Updates: this
• Clears: m
• Requires:

  \[ \text{DOMAIN}(\text{this}) \text{ intersection } \text{DOMAIN}(m) = \{\} \]

• Ensures:

  \[ \text{this} = \#\text{this union } \#m \]
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| ```c
m1 = {("PB", 99),
      ("BW", 17)}
m2 = {("PS", 99)}
``` | ```c
m1.combineWith(m2);
``` |
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*m1 = { (&quot;PB&quot;, 99),</td>
</tr>
<tr>
<td></td>
<td>(&quot;BW&quot;, 17) }</td>
</tr>
<tr>
<td></td>
<td>*m2 = { (&quot;PS&quot;, 99) }</td>
</tr>
<tr>
<td></td>
<td><strong>m1.combineWith(m2);</strong></td>
</tr>
<tr>
<td></td>
<td>*m1 = { (&quot;PB&quot;, 99),</td>
</tr>
<tr>
<td></td>
<td>(&quot;BW&quot;, 17),</td>
</tr>
<tr>
<td></td>
<td>(&quot;PS&quot;, 99) }</td>
</tr>
<tr>
<td></td>
<td><strong>m2 = { }</strong></td>
</tr>
</tbody>
</table>
sharesKeyWith

boolean sharesKeyWith(Map<K,V> m)

• Reports whether this and m have any keys in common.

• Ensures:

sharesKeyWith =

    (DOMAIN(this) intersection DOMAIN(m) /= {})
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>`m1 = { (&quot;PB&quot;, 99),</td>
<td>`m1 = { (&quot;PB&quot;, 99),</td>
</tr>
<tr>
<td>(&quot;BW&quot;, 17) }`</td>
<td>(&quot;BW&quot;, 17) }</td>
</tr>
<tr>
<td><code>m2 = { (&quot;PS&quot;, 99) }</code></td>
<td><code>m2 = { (&quot;PS&quot;, 99) }</code></td>
</tr>
<tr>
<td><code>boolean b =</code></td>
<td><code>boolean b =</code></td>
</tr>
<tr>
<td><code>m1.sharesKeyWith(m2);</code></td>
<td><code>m1.sharesKeyWith(m2);</code></td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| `m1 = {("PB", 99),
       ("BW", 17)}
m2 = {("PS", 99)}
     
     boolean b =
     m1.sharesKeyWith(m2);` | `m1 = {("PB", 99),
        ("BW", 17)}
m2 = {("PS", 99)}
b = false` |
iterator

Iterator<Map.Pair<K,V>> iterator()

• Returns an iterator over a set of elements of type Map.Pair<K,V>.

• Ensures:

\[
\text{entries}(\sim\text{this.seen} \times \sim\text{this.unseen}) = \text{this}
\]

and

\[
|\sim\text{this.seen} \times \sim\text{this.unseen}| = |\text{this}|
\]
Example

• Suppose you have a Map that keeps track of the names and associated salaries of all employees in the company:

\[
\text{Map<String,NaturalNumber> m = new Map1L<>();}
\]

\[
\ldots
\]
Sample For-Each Loop: Danger!

• Here’s how you *might try* to give every employee a $10,000 raise:

```java
NaturalNumber raise = new NaturalNumber2(10000);
for (Map.Pair<String, NaturalNumber> p : m) {
    NaturalNumber salary = p.value();
    salary.add(raise);
}
```
Sample For-Each Loop:

• Here's how you might try to give every employee a $10,000 raise:

```java
NaturalNumber raise = new NaturalNumber2(10000);
for (Map.Pair<String,NaturalNumber> p : m) {
    NaturalNumber salary = p.value();
    salary.add(raise);
}
```

Draw this diagram: `p` holds aliases to some key and its associated value in `m`; the method `value` returns an alias to a `NaturalNumber` that is also in the `Map m`; so, changing that `NaturalNumber` incidentally changes the values of both `p` and `m` (even though no method is called in the loop).
Sample For-Each Loop: Danger!

- Here’s how you might try to give every employee a $10,000 raise:

```java
NaturalNumber raise =
    new NaturalNumber2(10000);
for (Map.Pair<String, NaturalNumber> p : m) {
    NaturalNumber salary = p.value();
    salary.add(raise);
}
```

**Danger!**

This violates the rules for using iterators and for-each loops!
The Safe Way

Here’s how you should give every employee a $10,000 raise:

```
NaturalNumber raise = new NaturalNumber2(10000);
Map<String, NaturalNumber> temp = m.newInstance();
temp.transferFrom(m);
while (temp.size() > 0) {
    Map.Pair<String, NaturalNumber> p =
        temp.removeAny();
    p.value().add(raise);
    m.add(p.key(), p.value());
}
```
Here's how you should give every employee a $10,000 raise:

```
NaturalNumber raise = new NaturalNumber2(10000);
Map<String, NaturalNumber> temp = m.newInstance();
temp.transferFrom(m);
while (temp.size() > 0) {
    Map.Pair<String, NaturalNumber> p = temp.removeAny();
    p.value().add(raise);
    m.add(p.key(), p.value());
}
```

Draw this diagram: p holds references to some key and its associated value, but now they are not in any Map and p is not in any Map; the method `value` returns an alias to a `NaturalNumber` in the `Map.Pair` p; so, changing that `NaturalNumber` does not incidentally change the value of m or temp (even though that actually would be OK for this loop).
Resources

• OSU CSE Components API: Map
  - http://web.cse.ohio-state.edu/software/common/doc/