Map
Map

• The \textbf{Map} component family allows you to manipulate \textit{mappings} from \textit{keys} (of any type $K$) to \textit{values} (of any type $V$)
  – A \textbf{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 \texttt{Map<String, Integer>} variable
Interfaces and Classes

- **Standard**
- **Iterable**
- **MapKernel**
- **Map**
  - Map1L
  - Map2
  - Map3
Interfaces and Classes

Standard

Iterable

Kernel

Map

Map1L

Map2

Map3

Standard has contracts for three methods:
clear
newInstance
transferFrom
Interfaces and Classes

MapKernel has contracts for six methods:
- add
- remove
- removeAny
- value
- hasKey
- size

MapKernel extends
- Map
- Iterable

Map implements
- Map2

Map2 extends
- MapKernel
- Iterable

Map extends
- Standard
- Iterable
Map has contracts for five other methods:

- replaceValue
- key
- hasValue
- sharesKeyWith
- combineWith
Interfaces and Classes

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

Iterables have 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

\[
\text{PARTIAL\_FUNCTION \textit{is finite set of}}
\]
\[
(\text{key: } K, \text{ value: } V)
\]
\[
\text{exemplar } m
\]
\[
\text{constraint}
\]
\[
\text{for all } \text{key1, key2: } K, \text{ value1, value2: } V \quad \text{where} \quad ((\text{key1, value1}) \text{ is in } m \quad \text{and} \quad ((\text{key2, value2}) \text{ is in } m))
\]
\[
(\text{if } \text{key1} = \text{key2} \text{ then } \text{value1} = \text{value2})
\]
Partial Function

PARTIAL_FUNCTION is finite set of (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)

This formally states “the function property” for a set of ordered pairs.
Domain of a (Partial) Function

\[
\text{DOMAIN (}
\begin{align*}
m: & \text{ PARTIAL\_FUNCTION} \\
& \text{finite set of } K \\
& \text{satisfies} \\
& \text{for all } \text{key: } K \\
& (\text{key is in } \text{DOMAIN}(m) \iff \\
& \text{there exists } \text{value: } V \\
& ((\text{key, value) is in m}))
\end{align*}
\]
Range of a (Partial) Function

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

• Formally:

    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></td>
</tr>
<tr>
<td></td>
<td>m = { }</td>
</tr>
</tbody>
</table>
add

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>
<tbody>
<tr>
<td></td>
<td>( m = {(&quot;PB&quot;, 99), )</td>
</tr>
<tr>
<td></td>
<td>( (&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></td>
<td>( m.add(k, v); )</td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>Is the requires clause satisfied? What is ( \text{DOMAIN}(m) )?</th>
</tr>
</thead>
</table>

<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;PS&quot; )</td>
</tr>
<tr>
<td>( v = 99 )</td>
</tr>
</tbody>
</table>

\( m \text{.add}(k, v); \)
### 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);  |                         |
| m = { ("PB", 99),
      ("BW", 17),
      ("PS", 99) } |
| k = "PS"      | v = 99                  |
### 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>
</table>
| m = { ("PB", 99),
      ("BW", 17) } |
| k = "PS" |
| v = 99 |
```

```
| |
| m = { ("PB", 99),
      ("BW", 17),
      ("PS", 99) } |
| k = "PS" |
| v = 99 |
```
Another Interface

• The Map interface includes an interface for another related generic type, Map.Pair

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

• Formally:

  type Map.Pair is modeled by
  (key: K, value: V)
Map.Pair Methods

• This (immutable) type has only a constructor (taking a \( K \) and a \( V \)) and a \textbf{getter} method for each pair component

  - \( K \ \text{key}() \)
    • Returns the first component of \textit{this}
    • Aliases: reference returned by \textit{key}
  
  - \( V \ \text{value}() \)
    • Returns the second component of \textit{this}
    • Aliases: reference returned by \textit{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>
|     | \[ m = \{ ("PB", 99),
          ("BW", 17) \} \] |
| Map.Pair<String, Integer> p = m.remove(k); | \[ k = "BW" \] |
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
|                             | \[ m = \{("PB", 99),
           ("BW", 17)\} \] |
| `k = "BW"`                  |                                 |
| `Map.Pair<String, Integer> p = m.remove(k);` | \[ k = "BW" \] |
|                             | \[ m = \{("PB", 99)\} \] |
|                             | \[ k = "BW" \] |
|                             | \[ p = ("BW", 17) \] |
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>m = { (&quot;PB&quot;, 99), (&quot;BW&quot;, 17), (&quot;PS&quot;, 99) }</td>
<td>m = { (&quot;PB&quot;, 99), (&quot;BW&quot;, 17), (&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>
</table>
| \[
\text{Map.Pair<String, Integer> } p = \\
m.\text{removeAny();}
\] | \[
\begin{align*}
m &= \{("PB", 99), \\
&\quad ("BW", 17), \\
&\quad ("PS", 99)\}
\end{align*}
\] |
| \[
\begin{align*}
m &= \{("PB", 99), \\
&\quad ("BW", 17)\}
\end{align*}
\] | \[
\begin{align*}
p &= ("PS", 99)
\end{align*}
\] |
value

V value(K key)

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

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

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

```
13 April 2015 OSU CSE 33

m = {("PB", 99),
     ("BW", 17)}
k = "PB"
v = -423
```

```
13 April 2015 OSU CSE 33

m = {("PB", 99),
     ("BW", 17)}
k = "PB"
v = 99
```
hasKey

boolean hasKey(K key)

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

• Ensures:

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

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

```java
```
**Example**

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

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
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| m = {("PB", 99),
      ("BW", 17)}

k = "PB"

v = 85 |

Integer oldV =

m.replaceValue(k, v);
## 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>$\text{(&quot;BW&quot;, 17)}}$</td>
</tr>
<tr>
<td>k = &quot;PB&quot;</td>
<td>$k = &quot;PB&quot;$</td>
</tr>
<tr>
<td>v = 85</td>
<td>$v = 85$</td>
</tr>
<tr>
<td>Integer oldV =</td>
<td>$m = {(&quot;PB&quot;, 85),$</td>
</tr>
<tr>
<td>m.replaceAll(k, v);</td>
<td>$\text{(&quot;BW&quot;, 17)}}$</td>
</tr>
<tr>
<td></td>
<td>$k = &quot;PB&quot;$</td>
</tr>
<tr>
<td></td>
<td>$v = 85$</td>
</tr>
<tr>
<td></td>
<td>$\text{oldV} = 99$</td>
</tr>
</tbody>
</table>
### Example

**State**

| m = {("PB", 99),
|      ("BW", 17)}
| k = "PB"
| v = 85

oldV = \text{85}

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>
|                       | $m = \{("PB", 99),
|                       |    ("BW", 17)\}$                         |
|                       | $k = "PB"$                                 |
|                       | $v = 85$                                   |
| $v = m\text{.replaceValue}(k, v);$ |                                        |
Another 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>k = &quot;PB&quot;</td>
<td>k = &quot;PB&quot;</td>
</tr>
<tr>
<td>v = 85</td>
<td>v = 85</td>
</tr>
<tr>
<td>v = m.replaceValue(k, v);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m = { (&quot;PB&quot;, 85), (&quot;BW&quot;, 17) }</td>
</tr>
<tr>
<td></td>
<td>k = &quot;PB&quot;</td>
</tr>
<tr>
<td></td>
<td>v = 99</td>
</tr>
</tbody>
</table>
Another Example

This use of the method avoids creating an alias: it \textbf{swaps} \( v \) with the value in \( m \) that was previously associated with \( k \).

\begin{verbatim}
v = m.replaceValue(k, v);
\end{verbatim}

\begin{tabular}{|c|c|}
\hline
\textbf{State} & \\
\hline
\( m = \{ ("PB", 99), \\
       ("BW", 17) \} \) & \\
\hline
\( k = "PB" \) & \\
\hline
\( v = 85 \) & \\
\hline
\hline
\( m = \{ ("PB", 85), \\
       ("BW", 17) \} \) & \\
\hline
\( k = "PB" \) & \\
\hline
\( v = 99 \) & \\
\hline
\end{tabular}
key

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>
<tbody>
<tr>
<td>`m = {(&quot;PB&quot;, 99),</td>
<td><code>m = {(&quot;PB&quot;, 99), (</code>BW&quot;, 17)</td>
</tr>
<tr>
<td>(&quot;BW&quot;, 17)};</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td><code>k = &quot;xyz&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>v = 99</code></td>
</tr>
<tr>
<td><code>k = m.key(v);</code></td>
<td></td>
</tr>
</tbody>
</table>


### Example

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></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`. 

```java
m = {("PB", 99),
    ("BW", 17)}
k = "PB"
v = 99
```
hasValue

boolean hasValue(V value)

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

• Ensures:

  hasValue =

  (value is in RANGE(this))
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = {(&quot;PB&quot;, 99), \newline (&quot;BW&quot;, 17)}$</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>
</tbody>
</table>
Example

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

```java
b = true
```
### Code Example

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

| m = {("PB", 99),
|      ("BW", 17)} |
| v = 17 |
| b = **true** |

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**.
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:
  this = #this union #m
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( m_1 = { (&quot;PB&quot;, 99), (&quot;BW&quot;, 17) } )</td>
</tr>
<tr>
<td></td>
<td>( m_2 = { (&quot;PS&quot;, 99) } )</td>
</tr>
<tr>
<td>( m_1 \text{.combineWith}(m_2); )</td>
<td></td>
</tr>
</tbody>
</table>
### Example

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

boolean sharesKeyWith(Map<K,V> m)

• Reports whether \texttt{this} and \texttt{m} have any keys in common.
• Ensures:

\[
\text{sharesKeyWith} = (\text{DOMAIN}(\texttt{this}) \cap \text{DOMAIN}(\texttt{m})) \neq \{\}\)

13 April 2015 OSU CSE
### Example

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

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( m_1 = { (&quot;PB&quot;, 99), (&quot;BW&quot;, 17) } )</td>
</tr>
<tr>
<td>boolean ( b = )</td>
<td></td>
</tr>
<tr>
<td>( m_1 \text{.sharesKeyWith}(m_2) ; )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( m_2 = { (&quot;PS&quot;, 99) } )</td>
</tr>
<tr>
<td></td>
<td>( b = \text{false} )</td>
</tr>
</tbody>
</table>
iterator

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

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

- Ensures:

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

\[
\text{and}
\]

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

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

```java
Map<String, NaturalNumber> m =
    new Map1L<>();
...
```
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:
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 $\text{value}$ returns an alias to a $\text{NaturalNumber}$ that is also in the $\text{Map}$ $m$; so, changing that $\text{NaturalNumber}$ 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:

```
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:

```java
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());
}
```
The Safe Way

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

```java
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://cse.osu.edu/software/common/doc/