NaturalNumber
The `NaturalNumber` component family allows you to manipulate natural numbers (i.e., non-negative integers)

- Unlike an `int` variable, a `NaturalNumber` variable has no upper bound on its value
- On the other hand, you need to call methods to do arithmetic; there are no nice built-in operators (e.g., `+`, `-`, `*`, `==`, `<`, ...) or literals (e.g., `0`, `1`, `13`, ...) as with `int` variables
Interfaces and Classes

Standard has contracts for three methods:
- clear
- newInstance
- transferFrom

Standard extends NaturalNumber.

NaturalNumber implements NaturalNumber1L and NaturalNumber2.
Interfaces and Classes

```
NaturalNumberKernel
```

has contracts for three methods:
```
multiplyBy10
divideBy10
isZero
```
Interfaces and Classes

NaturalNumber has contracts for 14 other methods, e.g.,
add
subtract
etc.
The **Standard** Interface

- The interface **Standard** has methods that are part of most (nearly all) OSU CSE component families
  - Separating the **standard methods** into their own interface means that these highly reused methods are *described in exactly one place*
The Standard Interface

- The interface Standard methods are part of most (nearly all) OSU CSE component families
  - Separating the standard methods into their own interface means that these highly reused methods are described in exactly one place.

This design goal in software engineering is usually called single point of control over change.
The Kernel Interface

• The interface `NaturalNumberKernel` has a minimal set of methods that are primitive in the `NaturalNumber` component family
  – Separating these `kernel (primary) methods` into their own interface identifies them as special in this regard
The Kernel Interface

- The interface `NaturalNumberKernel` has a minimal set of methods that are *primitive* in the `NaturalNumber` component family.
  - Separating these *kernel (primary) methods* into their own interface identifies them as special in this regard.

The choice of kernel methods is a key decision by the designer of a component family.
The Enhanced Interface

- The interface `NaturalNumber` has all other methods that are *convenient to have* in the `NaturalNumber` component family
  - These *secondary methods* are often more “powerful” than the kernel methods and are introduced to make the component family readily usable in typical client code
Mathematical Model

• The value of a `NaturalNumber` variable is modeled as a non-negative `integer`

• Formally:

```
NATURAL  is  integer

exemplar  n

constraint  n  >=  0

type  NaturalNumber  is  modeled by

NATURAL
```
First, we define the mathematical model we intend to use, including any constraints that limit the values it might have.

The value of a NaturalNumber variable is modeled as a non-negative integer.

Formally:

\[ \text{NATURAL is integer} \]
\[ \text{exemplar } n \]
\[ \text{constraint } n \geq 0 \]
\[ \text{type NaturalNumber is modeled by } \text{NATURAL} \]
Mathematical Model

• The value of a `NaturalNumber` variable is modeled as a non-negative integer.

• Formally:

```plaintext
NATURAL is integer
example n
constraint n >= 0

type NaturalNumber is modeled by NATURAL
```

Second, we state that a `NaturalNumber` variable has that mathematical model.
Constructors

• There are four *constructors* for each implementation class

• As always:
  – The name of the constructor is the name of the implementation class
  – Constructors differ only in their parameters
  – Each has its own contract (which is in the kernel interface `NaturalNumberKernel`
No-argument Constructor

• A constructor with no parameters is called a **no-argument constructor**
• Ensures:

\[
\text{this} = 0
\]
<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaturalNumber n = new</td>
<td></td>
</tr>
<tr>
<td>NaturalNumber2();</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>State</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>NaturalNumber n = new NaturalNumber2();</code></td>
<td><code>n = 0</code></td>
</tr>
</tbody>
</table>
Copy Constructor

• There is a constructor with one parameter of the same type (\texttt{NaturalNumber n}), and it returns a copy of the parameter value so it is called a \textit{copy constructor}

• Ensures:
  \[ \texttt{this} = n \]
### Example

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>k = 12345678909</code></td>
<td></td>
</tr>
<tr>
<td><code>NaturalNumber m = new NaturalNumber2(k);</code></td>
<td></td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 12345678909$</td>
<td>$k = 12345678909$</td>
</tr>
<tr>
<td><code>NaturalNumber m = new NaturalNumber2(k);</code></td>
<td>$m = 12345678909$</td>
</tr>
</tbody>
</table>
Constructor from `int`

- There is a constructor with one parameter `int i`
- Requires: $i \geq 0$
- Ensures: `this = i`
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaturalNumber n = new NaturalNumber2(j);</td>
<td>( j = 13 )</td>
</tr>
</tbody>
</table>
# Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>j = 13</strong></td>
<td><strong>j = 13</strong></td>
</tr>
<tr>
<td><code>NaturalNumber n = new NaturalNumber2(j);</code></td>
<td></td>
</tr>
</tbody>
</table>
| **j = 13**                                | **j = 13**
| **n = 13**                                | **n = 13** |
Constructor from String

• There is a constructor with one parameter
  String s

• Requires:
  \( \text{there exists } n : \text{NATURAL} \)
  \[ (s = \text{TO\_STRING}(n)) \]

• Ensures:
  \[ s = \text{TO\_STRING}(\text{this}) \]
Constructor from String

• There is a constructor

```java
String s
```

• Requires:

```java
there exists n: NATURAL
(s = TO_STRING(n))
```

• Ensures:

```java
s = TO_STRING(this)
```

In other words, `s` must look like the result of converting some NaturalNumber value to a String...
Constructor from `String`

- There is a constructor:
  
  ```java
  String s
  ```

- Requires:
  
  ```java
  there exists n: NATURAL
  \( (s = \text{TO\_STRING}(n)) \)
  ```

- Ensures:
  
  ```java
  s = \text{TO\_STRING}(\text{this})
  ```

... and the `NaturalNumber` value resulting from the constructor is what would have given you that `String`. 
**Example**

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>s = &quot;265&quot;</td>
<td></td>
</tr>
<tr>
<td>NaturalNumber n = new NaturalNumber2(s);</td>
<td></td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>s = &quot;265&quot;</td>
<td>s = &quot;265&quot;</td>
</tr>
<tr>
<td>NaturalNumber n = new</td>
<td>n = 265</td>
</tr>
<tr>
<td>NaturalNumber2(s);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s = &quot;265&quot;</td>
</tr>
</tbody>
</table>
Methods for \texttt{NaturalNumber}

• All the methods for \texttt{NaturalNumber} are \textit{instance methods}, i.e., you call them as follows:

\begin{center}
\texttt{n.methodName(arguments)}
\end{center}

where \texttt{n} is an initialized variable of type \texttt{NaturalNumber}
Methods for **NaturalNumber**

- All the methods for **NaturalNumber** are *instance methods*, i.e., you call them as follows:

  \[ \text{n.methodName(arguments)} \]

  where `n` is an initialized variable of type **NaturalNumber**.

  Recall: `n` is called the *receiver*; for all instance methods, the corresponding *distinguished formal parameter* implicitly has the name `this`. 
Order of Presentation

• The methods are introduced here starting with those you might expect to see as a client, and then proceeding to ones that might seem more surprising

• Methods not discussed here:
  - `setFromInt`, `canConvertToInt`, `toInt`
  - `setFromString`, `canSetFromString`
  - `increment`, `decrement`
void add(NaturalNumber n)

• Adds $n$ to this.
• Updates: this
• Ensures:

\[
\text{this} = \#\text{this} + n
\]
add

```java
void add(NaturalNumber n)

• Adds n to this.
• Updates: this
• Ensures:
  this = #this
```

The parameter mode called `updates` in a contract means the variable’s value `might be changed` by a call to the method.
**add**

```java
void add(NaturalNumber n)
• Adds n to this.
• Updates: this
• Ensures:
  this = #this
```

If *this* is an *updates-mode parameter* in any method, then the type in question is *mutable*.
void add(NaturalNumber n)

• Adds \( n \) to \( \text{this} \).
• Updates: \( \text{this} \)
• Ensures:
  \[
  \text{this} = \#\text{this} + n
  \]

In an ensures clause, a \( \# \) in front of a variable whose value might be changed is pronounced “old”; \( \#\text{this} \) denotes the old, or incoming, value of \( \text{this} \).
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 143</td>
<td>m = 143</td>
</tr>
<tr>
<td>k = 70</td>
<td>k = 70</td>
</tr>
<tr>
<td>m.add(k);</td>
<td></td>
</tr>
</tbody>
</table>

---

7 January 2019   OSU CSE
### Example

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></th>
</tr>
</thead>
</table>
|           | $m = 143$  
|           | $k = 70$   |
| `m.add(k);` |           |
|           | $m = 213$  
|           | $k = 70$   |
**void** subtract(NaturalNumber n)

- Subtracts `n` from `this`.
- Updates: `this`
- Requires:
  
  ```
  this >\geq n
  ```
- Ensures:
  
  ```
  this = \#this - n
  ```
void subtract(NaturalNumber n)
• Subtracts \( n \) from \( this \).
• Updates: \( this \)
• Requires: \( this \geq n \)
• Ensures: \( this = \#this - n \)

Important! It could have been written as:
\[ \#this = this + n \]
subtract

\textbf{void} subtract(NaturalNumber n)

- Subtracts \( n \) from \textit{this}.
- Updates: \textit{this}
- Requires: \( \textit{this} \geq n \)
- Ensures:
  \( \textit{this} = \#\textit{this} - n \)

Or even as:
\( \textit{this} + n = \#\textit{this} \)
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = 143$</td>
<td></td>
</tr>
<tr>
<td>$k = 70$</td>
<td></td>
</tr>
<tr>
<td>$m.subtract(k);$</td>
<td></td>
</tr>
</tbody>
</table>

The code snippet `m.subtract(k);` modifies the value of `m` by subtracting `k`. The initial values of `m` and `k` are 143 and 70, respectively.
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = 143$</td>
<td>$m = 73$</td>
</tr>
<tr>
<td>$k = 70$</td>
<td>$k = 70$</td>
</tr>
<tr>
<td>$m \text{.subtract}(k)$;</td>
<td></td>
</tr>
</tbody>
</table>
void multiply(NaturalNumber n)

• Multiplies this by n.
• Updates: this
• Ensures:

\[
\text{this} = \#\text{this} \times n
\]
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = 143$</td>
</tr>
<tr>
<td></td>
<td>$k = 70$</td>
</tr>
<tr>
<td>m.multiply(k);</td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 143</td>
<td>k = 70</td>
</tr>
<tr>
<td>m.multiply(k);</td>
<td></td>
</tr>
<tr>
<td>m = 10010</td>
<td>k = 70</td>
</tr>
</tbody>
</table>
divide

NaturalNumber divide(NaturalNumber n)

• Divides this by \( n \), returning the remainder.
• Updates: this
• Requires: 
  \( n > 0 \)
• Ensures:
  \[
  \#this = n \ast this + divide \quad \text{and} \\
  0 \leq divide < n
  \]
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NaturalNumber r = m.divide(k);</code></td>
<td><code>m = 143</code></td>
</tr>
<tr>
<td></td>
<td><code>k = 70</code></td>
</tr>
</tbody>
</table>
# Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = 143$</td>
<td>$m = 143$</td>
</tr>
<tr>
<td>$k = 70$</td>
<td>$k = 70$</td>
</tr>
<tr>
<td><code>NaturalNumber r = m.divide(k);</code></td>
<td></td>
</tr>
<tr>
<td>$m = 2$</td>
<td>$m = 2$</td>
</tr>
<tr>
<td>$k = 70$</td>
<td>$k = 70$</td>
</tr>
<tr>
<td>$r = 3$</td>
<td>$r = 3$</td>
</tr>
</tbody>
</table>
void power(int p)

• Raises this to the power p.

• Updates: this

• Requires:
    
    \[ p \geq 0 \]

• Ensures:
    
    \[ this = #this ^ (p) \]
void power(int p)

- Raises \texttt{this} to the power \texttt{p}.
- Updates: \texttt{this}
- Requires: \texttt{p} \geq 0
- Ensures: \texttt{this} = \texttt{\#this} ^ (p)

Note: 0 ^ (0) = 1 by definition of the ^ operator.
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m.power(k);</code></td>
<td>$m = 143$</td>
</tr>
<tr>
<td></td>
<td>$k = 4$</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m.power(k);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( m = 143 )</td>
</tr>
<tr>
<td></td>
<td>( k = 4 )</td>
</tr>
<tr>
<td></td>
<td>( m = 418161601 )</td>
</tr>
<tr>
<td></td>
<td>( k = 4 )</td>
</tr>
</tbody>
</table>
void root(int r)

• Updates this to the $r$-th root of its incoming value.

• Updates: this

• Requires:
  $r \geq 2$

• Ensures:
  
  $this^{(r)} \leq #this < (this + 1)^{(r)}$
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m.root(k);</td>
<td>$m = 143$</td>
</tr>
<tr>
<td></td>
<td>$k = 2$</td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| m.root(k);    | $m = 143$
|               | $k = 2$  |
|               | $m = 11$
|               | $k = 2$  |
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m.root(k);</td>
<td>m = 144</td>
</tr>
<tr>
<td></td>
<td>k = 2</td>
</tr>
<tr>
<td>m = 12</td>
<td></td>
</tr>
<tr>
<td>k = 2</td>
<td></td>
</tr>
</tbody>
</table>
void copyFrom(NaturalNumber n)

• Copies \( n \) to \texttt{this}.
• Replaces: \texttt{this}
• Ensures:

\[
\texttt{this} = n
\]
void copyFrom(NaturalNumber n)

• Copies \( n \) to \( \text{this} \).

• Replaces: \( \text{this} \)

• Ensures:
  \[
  \text{this} = n
  \]

The parameter mode called replaces in a contract means the variable’s value might be changed by a call to the method, but the new value is independent of the old value.
`copyFrom`

```java
void copyFrom(NaturalNumber n)
```

- Copies `n` to `this`.
- Replaces: `this`
- Ensures:
  ```java
  this = n
  ```

If `this` is a `replaces-mode parameter` in any method, then the type in question is `mutable`. 
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m.copyFrom(k);</td>
<td>$m = 143$</td>
</tr>
<tr>
<td></td>
<td>$k = 70$</td>
</tr>
</tbody>
</table>
**Example**

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = 143$</td>
</tr>
<tr>
<td></td>
<td>$k = 70$</td>
</tr>
<tr>
<td><code>m.copyFrom(k);</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$m = 70$</td>
</tr>
<tr>
<td></td>
<td>$k = 70$</td>
</tr>
</tbody>
</table>
compareTo

int compareTo(NaturalNumber n)

• Compares \texttt{n} to \texttt{this}, returning a negative number if \texttt{this} < \texttt{n}, 0 if \texttt{this} = \texttt{n}, and a positive number if \texttt{this} > \texttt{n}

• Ensures:

\[
\text{compareTo} = \text{[a negative number, zero, or a positive integer as this is less than, equal to, or greater than n]}
\]
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{int} comp =</td>
<td>\textit{m} = 143</td>
</tr>
<tr>
<td>\hspace{1em} m.compareTo(k);</td>
<td>\textit{k} = 70</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 143</td>
<td>m = 143</td>
</tr>
<tr>
<td>k = 70</td>
<td>k = 70</td>
</tr>
<tr>
<td>int comp = m.compareTo(k);</td>
<td>comp = 1</td>
</tr>
</tbody>
</table>

m = 143
k = 70
comp = 1
Example

Though here the result of the method is 1, it could be any positive int, so don’t assume it is 1.

```
int comp = m.compareTo(k);
```

State

```
m = 143
k = 70
comp = 1
```
multiplyBy10

void multiplyBy10(int k)

• Multiplies this by 10 and adds k.

• Updates: this

• Requires:

\[ 0 \leq k < 10 \]

• Ensures:

\[ \text{this} = 10 \times \#\text{this} + k \]
void multiplyBy10(int k)

• Multiplies this by 10 and adds

• Updates: this

• Requires:

  \[ 0 \leq k < 10 \]

• Ensures:

  \[ this = 10 \times \#this + k \]
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( m = 143 )</td>
</tr>
<tr>
<td></td>
<td>( d = 7 )</td>
</tr>
<tr>
<td>\texttt{m.multiplyBy10(d);}</td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m.multiplyBy10(d);</td>
<td>m = 143</td>
</tr>
<tr>
<td></td>
<td>d = 7</td>
</tr>
<tr>
<td></td>
<td>m = 1437</td>
</tr>
<tr>
<td></td>
<td>d = 7</td>
</tr>
</tbody>
</table>
divideBy10

int divideBy10()

• Divides this by 10 and returns the remainder.
• Updates: this
• Ensures:

#this = 10 * this + divideBy10 and
0 <= divideBy10 < 10
**divideBy10**

```c
int divideBy10() {
    // Divides this by 10 and returns the remainder.
    // Updates: this
    // Ensures:
    #this = 10 * this + divideBy10 and 0 <= divideBy10 < 10
}
```

This is a kernel method.
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| \textbf{int } r = \\
\text{ m.divideBy10();} | $m = 1437$ |
# Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m = 1437$</td>
</tr>
<tr>
<td>int $r = m$.divideBy10();</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$m = 143$</td>
</tr>
<tr>
<td></td>
<td>$r = 7$</td>
</tr>
</tbody>
</table>
boolean isZero()

• Reports whether this is zero.
• Ensures:

   isZero = (this = 0)
isZero

boolean isZero()

• Reports whether this is zero.
• Ensures:

  \[ isZero = (this = 0) \]

This is a kernel method.
<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( m = 143 )</td>
</tr>
<tr>
<td>\texttt{boolean}\ ( z = ) \texttt{m.isZero();}</td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = 143$</td>
<td>$m = 143$</td>
</tr>
<tr>
<td><code>boolean z = m.isZero();</code></td>
<td>$z = \text{false}$</td>
</tr>
</tbody>
</table>
void clear()

• Resets this to an initial value.
• Clears: this
• Ensures:

  this = 0
clear

void clear()

• Resets \texttt{this} to an initial value.
• Clears: \texttt{this}
• Ensures:
  \[
  \texttt{this} = 0
  \]

This is a standard method.
clear

```java
void clear()

• Resets this to an initial value.
• Clears: this
• Ensures: 
  this = 0
```

The *parameter mode* called `clears` in a contract means the variable’s value is reset to an initial value by a call to the method.
clear

```java
void clear()
```

- Resets `this` to an initial value.
- Clears: `this`
- Ensures:
  ```
  this = 0
  ```

If `this` is a clears-mode parameter in any method, then the type in question is `mutable`. 
clear

```c
void clear()

• Resets this to an initial value.
• Clears: this
• Ensures: this = 0
```

The ensures clause is redundant in this case because this is a clears-mode parameter.
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m.clear();</td>
<td>$m = 143$</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 143</td>
<td>m = 143</td>
</tr>
<tr>
<td>m.clear();</td>
<td></td>
</tr>
<tr>
<td></td>
<td>m = 0</td>
</tr>
</tbody>
</table>
newInstance

NaturalNumber newInstance()

• Returns a new object with the same implementation as this, having an initial value.

• Ensures:
  \[ \text{newInstance} = 0 \]
newInstance

NaturalNumber newInstance()  

• Returns a new object with the same implementation as \texttt{this}, having an initial value.  

• Ensures:  

\[ newInstance = 0 \]  

This is a standard method.
newInstance

NaturalNumber newInstance()

• Returns a new object with the same implementation as this, having an initial value.
• Ensures:
  \[ \text{newInstance} = 0 \]

This is similar to a constructor; the difference is that you don’t need to know the name of any implementation class to call this method.
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 143</td>
<td>m = 143</td>
</tr>
<tr>
<td>NaturalNumber k = m.newInstance();</td>
<td></td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 143</td>
<td>m = 143</td>
</tr>
<tr>
<td>NaturalNumber k = m.newInstance();</td>
<td>k = 0</td>
</tr>
</tbody>
</table>
void transferFrom(NaturalNumber n)

- Sets \textit{this} to the incoming value of \textit{n}, and resets \textit{n} to an initial value; \textit{n} must be of the same implementation as \textit{this}.

- Replaces: \textit{this}

- Clears: \textit{n}

- Ensures:

  \textit{this} = \#\textit{n}
transferFrom

void transferFrom(NaturalNumber n)

• Sets this to the incoming value of n, and resets n to an initial value; n must be of the same implementation as this.
• Replaces: this
• Clears: n
• Ensures:
  \[ \text{this} = \#n \]

This is a standard method.
transferFrom

```java
void transferFrom(NaturalNumber n)
```

- Sets `this` to the incoming value of `n`, and resets `n` to an initial value; `n` must be of the same implementation as `this`.
- Replaces: `this`
- Clears: `n`
- Ensures: `this = #n`

This is similar to `copyFrom` but is *always more efficient*, so it should be used if you don’t really need a duplicate.
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m.transferFrom(k);</code></td>
<td><code>m = 143</code></td>
</tr>
<tr>
<td></td>
<td><code>k = 70</code></td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| `m.transferFrom(k);`       | \( m = 143 \)  
                          | \( k = 70 \)  |
|                             | \( m = 70 \)  
                          | \( k = 0 \)  |
Whoa! It Clears n?

- Did you notice that `transferFrom` changes the value of its argument? How can it do this? Didn’t we say that this can’t happen?
  - It can’t for arguments of Java’s primitive types

- There is a crucial difference between Java’s primitive types and all other types, that allows this behavior for other types
  - Details coming soon...
toString

String toString()

• Returns the string representation of this.
• Ensures:

    toString = [the string representation of this]
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>m = 143</code></td>
<td><code>m = 143</code></td>
</tr>
<tr>
<td><code>String s = m.toString();</code></td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| ```java
m = 143
String s =
m.toString();
``` | ```
m = 143
s = "143"
``` |
Resources

• OSU CSE Components API: NaturalNumber
  – http://cse.osu.edu/software/common/doc/