SortingMachine
SortingMachine

• The *SortingMachine* component family allows you to add elements of type $T$ to a collection of such elements, and then to remove them one at a time in *sorted order* according to a client-supplied ordering
  
  – *Queue* and *Stack* support removal in FIFO and LIFO order, respectively
The *SortingMachine* component family allows you to add elements of type $T$ to a collection of such elements, and then to remove them one at a time in *sorted order* according to a client-supplied ordering.

- Queue and Stack support removal in FIFO and LIFO order, respectively.

FIFO and LIFO are *time-based* orderings; a *SortingMachine* uses a *value-based* ordering.
Why Not Use The sort Method?

```java
while (/*input values remain*/) {
    /* let x = next input value */
    q.enqueue(x);
}
q.sort(order);
while (q.length() > 0) {
    T x = q.dequeue();
    /* output x */
}
```
Why Not Use The `sort` Method?

```java
while (/*input values in main*/) {
    /* let x = next input value */
    q.enqueue(x);
}
q.sort(order);
while (q.length() > 0) {
    T x = q.dequeue();
    /* output x */
}
```

The Java libraries have similar methods to sort built-in arrays and other "collections".
Example

• Suppose you want to find the students with the 50 highest GPAs among all Ohio State students
  – Modify the previous code to show how you might do this ...
Performance in the Example

• Code using a `sort` method spends time to sort all 50,000 GPAs just to find the top 50
• Code based on `SortingMachine` can be much more efficient, because if you don’t remove all of the elements you don’t necessarily pay for sorting all of them
Interfaces and Classes

- Standard
- Iterable
- SortingMachine-Kernel
- SortingMachine
- SortingMachine1L
- SortingMachine5
Interfaces and Classes

Standard

Iterable

SortingMachine-Kernel

extends

extends

implements

has contracts for six methods:
add
changeToExtractionMode
removeFirst
isInInsertionMode
order
size

...
There is really an abstract class as usual in the chain here, but it is not shown because these slides describe the client view, and a client needs only interface `SortingMachine` and a class like `SortingMachine1L`. 

```java
Interfaces and Classes

There is really an abstract class as usual in the chain here, but it is not shown because these slides describe the client view, and a client needs only interface `SortingMachine` and a class like `SortingMachine1L`.
```

```
```
**Mathematical Model**

\[\text{SORTING\_MACHINE\_MODEL} \text{ is } (\]
\[\text{insertion\_mode: boolean,}\]
\[\text{ordering: binary relation on } T,\]
\[\text{contents: finite multiset of } T\]
\[)\]

\textit{exemplar} \textit{m}

\textit{constraint}

\text{IS\_TOTAL\_PREORDER}(m.\text{ordering})

\textit{type} \text{SortingMachineKernel} \text{ is modeled by}

\text{SORTING\_MACHINE\_MODEL}
Mathematical Model

SORTING_MACHINE_MODEL is ( 
    insertion_mode: boolean,
    ordering: binary relation on T,
    contents: finite multiset of T
)

exemplar m

constraint
    IS_TOTAL_P

type SortingMachineKernel

SORTING_MACHINE_MODEL

The mathematical model is an ordered triple (a.k.a. three-tuple): a boolean, a binary relation on T, and a finite multiset of T.
Mathematical Model

SORTING_MACHINE_MODEL is (  
    insertion_mode: boolean,  
    ordering: binary relation on T,  
    contents: finite multiset of T  
)

exemplar m

constraint

IS_TOTAL_RELATION

type SortingMachineKernel

Recall: a binary relation on T may be viewed as a set of ordered pairs of T, or as a boolean-valued function R of two parameters of type T that is true iff that pair is in the set.
Mathematical Model

SORTING_MACHINE_MODEL is (  
  insertion_mode: boolean,  
  ordering: binary relation on T,  
  contents: finite multiset of T  
)

A finite multiset is essentially a finite set with multiple copies of elements allowed, so there are effectively (non-negative) “counts” of all values of the element type T; details as necessary.
Mathematical Model

| SORTING_MACHINE_MODEL is (           |
|   insertion_mode: boolean,         |
|   ordering: binary relation on T,  |
|   contents: finite multiset of T   |
| )                                 |
| exemplar m                        |
| constraint                        |
| IS_TOTAL_PREORDER(m.ordering)     |
| type SortingMachineKernel is modeled by |
| SORTING_MACHINE_MODEL             |
Review: Comparators

• The Java interface `Comparator<T>` is:

```java
public interface Comparator<T> {
    /**
     * Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.
     */
    int compare(T o1, T o2);
}
```
Review: Comparators

• The notion of “less than” and “greater than” can be anything

• The notion of “equal to” is actually supposed to be “equivalent to”, in the sense that the first argument is neither “less than” nor “greater than” the other

• There are important technicalities …
private static class IntegerLT implements Comparator<Integer> {
    @Override
    public int compare(Integer o1, Integer o2) {
        if (o1 < o2) {
            return -1;
        } else if (o1 > o2) {
            return 1;
        } else {
            return 0;
        }
    }
}
private static class IntegerLT implements Comparator<Integer> {
    @Override
    public int compare(Integer o1, Integer o2) {
        if (o1 < o2) {
            return -1;
        } else if (o1 > o2) {
            return 1;
        } else {
            return 0;
        }
    }
}

A class that implements Comparator is usually a nested class (i.e., declared for local use inside another class), and if so should be declared private static.
Review: Creating a Comparator

```java
private static class IntegerLT
    implements Comparator<Integer> {
    @Override
    public int compare(Integer o1, Integer o2) {
        return o1 - o2;
    }
}
```

Note that the results are not specified to be \(-1\) for “less than” and \(+1\) for “greater than”, but merely negative and positive values, respectively! Does this code work?
Review: An Easy Comparator

```java
private static class IntegerLT
    implements Comparator<Integer> {
    @Override
    public int compare(Integer o1, Integer o2) {
        return o1.compareTo(o2);
    }
}
```

Since a generic parameter must be a reference type, and each wrapper type T (here, Integer) implements the interface Comparable<T>, each has a compareTo method that can be called like this; it simplifies the code for compare from the previous Comparator<T> implementation (using <), if \( \leq \) is the mathematical ordering we want.
SortingMachine Constructor

• The constructor has one parameter `order` of type `Comparator<T>`
  – Sorting is based on the ordering provided by the `compare` method from `order`

• Requires:
  ```
  IS_TOTAL_PREORDER(order)
  ```

• Ensures:
  ```
  this = (true, order, {}) 
  ```
## Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| ```java
Comparator<Integer> ci =
    new IntegerLT ();
SortingMachine<Integer> si =
    new SortingMachine1L<>()(ci);
``` |   |
### Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Comparator&lt;Integer&gt; ci = new IntegerLT ();</code></td>
<td></td>
</tr>
<tr>
<td><code>SortingMachine&lt;Integer&gt; si = new SortingMachine1LL&lt;&gt;(ci);</code></td>
<td></td>
</tr>
<tr>
<td><code>si = (true, ci, {})</code></td>
<td></td>
</tr>
</tbody>
</table>
void add(T x)

• Adds x to the contents of this.
• Aliases: reference x
• Updates: this.contents
• Requires: this.insertion_mode
• Ensures:
  this.contents =
  #this.contents union {x}
void add(T x)

- Adds \( x \) to the contents of this.
- Aliases: reference to \( x \)
- Updates: this.contents
- Requires: this.insertion_mode
- Ensures:
  \[
  \text{this.contents} = \text{#this.contents} \ union \ \{x\}
  \]
### Example

<table>
<thead>
<tr>
<th><strong>Code</strong></th>
<th><strong>State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>$si = (\text{true}, ci, {13, 8})$</td>
<td>$x = 13$</td>
</tr>
<tr>
<td>$si.add(x);$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
</table>
| `si = (true, ci, {13, 8})`  
  `x = 13`  
  `si.add(x);` |                                           |
| `si = (true, ci, {13, 13, 8})`  
  `x = 13` |                                           |
Example

Note the alias created here, which you cannot see in the tracing table.

<table>
<thead>
<tr>
<th>State</th>
<th>si = (true, ci, {13, 8})</th>
<th>x = 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>si.add(x);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>si = (true, ci, {13, 13, 8})</td>
<td>x = 13</td>
<td></td>
</tr>
</tbody>
</table>
```cpp
void changeToExtractionMode()

• Change the mode of \texttt{this} from insertion to extraction.

• Updates: \texttt{this.insertion\_mode}

• Requires: 
  \texttt{this.insertion\_mode}

• Ensures: 
  \texttt{not this.insertion\_mode}
```
Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$si = (true, ci, {13, 13, 8})$</td>
</tr>
<tr>
<td>$si.changeToExtractionMode();$</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><code>si = (true, ci, {13, 13, 8})</code></td>
<td><code>si = (true, ci, {13, 13, 8})</code></td>
</tr>
<tr>
<td><code>si.changeToExtractionMode();</code></td>
<td><code>si = (false, ci, {13, 13, 8})</code></td>
</tr>
</tbody>
</table>
removeFirst

T removeFirst()

• Removes and returns some “first” (“smallest”) element from the contents of this.
• Updates: this.contents
• Requires:
  
  not this.insertion_mode and 
  this.contents /= {} 

• Ensures:

  removeFirst is in #this.contents and 
  for all x: T where (x is in this.contents) 
  (this.ordering(removeFirst, x)) and 
  this.contents = #this.contents \ {removeFirst}
**removeFirst**

T removeFirst()

- Removes and returns some "first" ("smallest") element from the contents of this.
- Updates: **this.contents**
- Requires: 
  
  - **not this.insertion_mode**
  - **this.contents /= {}**
- Ensures:
  
  - removeFirst *is in* **this.contents** and for all x: T where (x *is in* **this.contents**) 
    
    (this.ordering(removeFirst, x)) and **this.contents** = **#this.contents** \ {removeFirst}

For multisets (like **this.contents**), *is in* means the “count” of the given value is positive, i.e., that value is an element of the multiset.
removeFirst

T removeFirst()

• Removes and returns some “first” (“smallest”) element from the contents of this.

• Updates: this

• Requires:
  not this.insertion_mode and this.contents /= {}

• Ensures:
  removeFirst is in #this.contents and for all x: T where (x is in this.contents) (
    this.ordering(removeFirst, x)) and this.contents = #this.contents \ {removeFirst}

For multisets (like this.contents), \ means the “counts” of all values are subtracted (but all remain non-negative).

Example

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>si = (false, ci, {13, 13, 8}) x = -425</td>
<td>x = si.removeFirst();</td>
</tr>
</tbody>
</table>

12 February 2014

OSU CSE
**Example**

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$si = (false, ci, {13, 13, 8})$</td>
</tr>
<tr>
<td></td>
<td>$x = -425$</td>
</tr>
<tr>
<td>$x = si.removeFirst();$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$si = (false, ci, {13, 13})$</td>
</tr>
<tr>
<td></td>
<td>$x = 8$</td>
</tr>
</tbody>
</table>
The element removed is a “first” element based on the `IntegerLT` ordering, i.e., a smallest integer value.

<table>
<thead>
<tr>
<th>State</th>
<th>si = (false, ci, {13, 13, 8})</th>
<th>x = -425</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = si.removeFirst()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>si = (false, ci, {13, 13})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x = 8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Remove Another

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>si = (false, ci,</td>
<td>si = (false, ci,</td>
</tr>
<tr>
<td>{13, 13})</td>
<td>{13, 13})</td>
</tr>
<tr>
<td>x = 8</td>
<td>x = 8</td>
</tr>
<tr>
<td>x = si.removeFirst();</td>
<td></td>
</tr>
</tbody>
</table>
Example: Remove Another

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$si = (false, ci, {13, 13})$</td>
</tr>
<tr>
<td></td>
<td>$x = 8$</td>
</tr>
<tr>
<td>$x = si.removeFirst();$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$si = (false, ci, {13})$</td>
</tr>
<tr>
<td></td>
<td>$x = 13$</td>
</tr>
</tbody>
</table>
Example: Remove Another

The element removed is a “first” element based on the `IntegerLT` ordering, i.e., a smallest integer value; another such element remains!
isInInsertionMode

boolean isInInsertionMode()

• Reports whether this is in insertion mode.
• Ensures:

    isInInsertionMode = this.insertion_mode
Comparator<T> order() 

• Reports Comparator<T> being used for sorting by this.

• Ensures:

  order = this.ordering
size

int size()

• Reports the number of elements in this.contents.

• Ensures:

  \[ \text{size} = |\text{this.contents}| \]
int size() {
• Reports the number of elements in this.contents.
• Ensures:
  \[ \text{size} = |\text{this.contents}| \]
}

For multisets (like this.contents), \(|\cdot|\) means the sum of the multiset’s “counts” of all values.
iterator

Iterator<T> iterator()

• Returns an iterator over a multiset of elements of type T.

• Ensures:

\[ \text{multiset\_entries}(\sim\text{this.seen} \ast \sim\text{this.unseen}) = \text{this.contents} \]
iterator

Iterator<T>

• Returns an iterator over a set of elements of type T.

• Ensures:

\[
\text{multiset}_{\text{entries}} \left( \sim \text{this}.\text{seen} \ast \sim \text{this}.\text{unseen} \right) = \text{this}.\text{contents}
\]

\text{multiset}_{\text{entries}} \text{ is like entries for a string, except that each string entry’s “count” in the multiset is the same as its (occurrence) count in the string.}
Temporal Flexibility

- The contract for `SortingMachineKernel` does not tell the client when "sorting" occurs, so an implementation of `SortingMachineKernel` may pay the cost of comparing elements to each other:
  - During `add`
  - During `changeToExtractionMode`
  - During `removeFirst`
Instead of `sort` ...

```java
while(/*input values remain*/) { 
    /* let x = next input value */
    q.enqueue(x);
}
q.sort(order);
while(q.length() > 0) { 
    T x = q.dequeue();
    /* output x */
}
```
Instead of `sort` ...

```java
while (/*input values remain*/) {
    /* let x = next input value */
    sorter.add(x);
}

sorter.changeToExtractionMode();

while (sorter.size() > 0) {
    T x = sorter.removeFirst();
    /* output x */
}
```
Instead of \texttt{sort} ...

\begin{verbatim}
while (/*input values remain*/) {
    /* let x = next input value */
    sorter.add(x);
}

sorter.changeToExtractionMode();
for (int i = 0; i < 50; i++) {
    T x = sorter.removeFirst();
    /* output x */
}
\end{verbatim}

This is how you might remove only the “first 50” (“smallest 50”) values that were added to \texttt{sorter}. 
Why Is This Better Than `sort`?

- If all elements are *already sorted* by the end of `changeToExtractionMode`, then there is no difference in performance compared to using `sort`.

- If all elements are *not already sorted* by the end of `changeToExtractionMode`, then there can be a performance advantage if you don’t need to remove all the elements!
  
  – See a future project ...
Sorting Algorithms

• Implementer of `SortingMachineKernel` may embed any *sorting algorithm* in it, e.g.:
  – *insertion sort* (*add* does all the work)
  – *quicksort* (*changeToExtractionMode* does all the work)
  – *selection sort* (*removeFirst* does all the work)
• Implementer of SortingMachineKernel may embed any sorting algorithm in it, e.g.:
  – insertion sort (add does all the work)
  – quicksort (changeToExtractionMode does all the work)
  – selection sort (removeFirst does all the work)

Actually, any sorting algorithm could be used when sorting is done entirely in changeToExtractionMode.
Resources

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