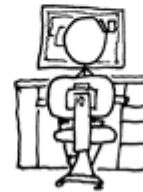
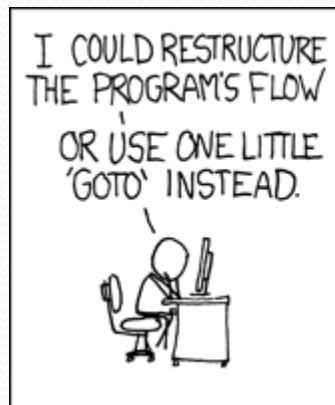


CSE 143

Lecture 3: `ArrayIntList`;
pre/post conditions and exceptions

reading: 4.4 15.1 - 15.3

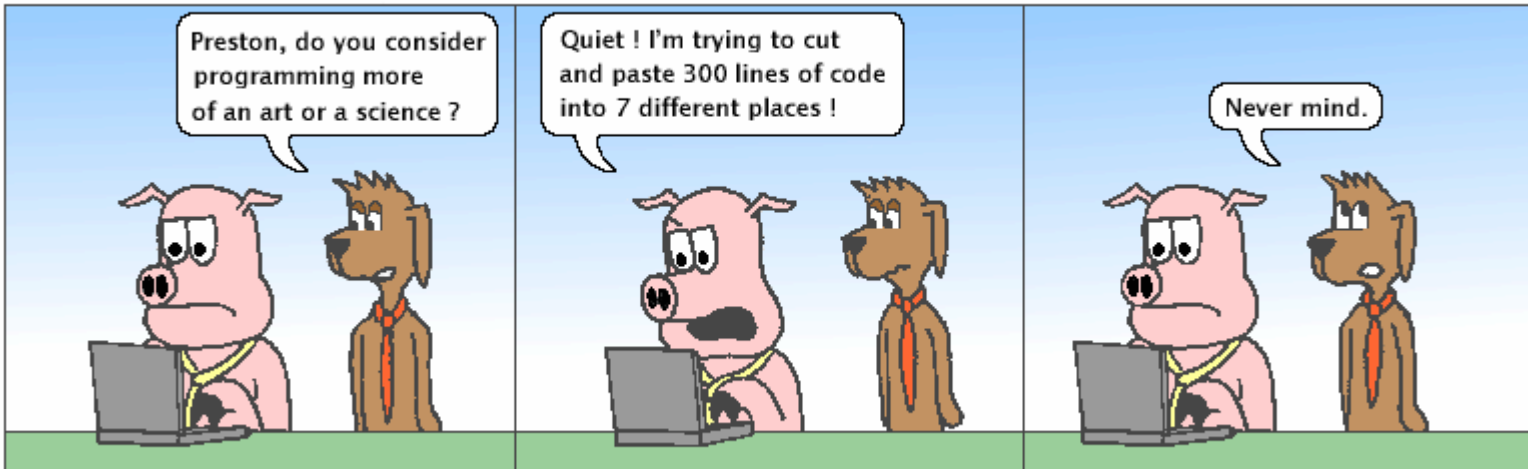


Why does style matter?

- Maintenance
 - `// magic number`
`int magicNumber = 9;`
- Getting a job
 - Every company has a different style guide

Hackles

By Drake Emko & Jen Brodzik



<http://hackles.org>


Copyright © 2001 Drake Emko & Jen Brodzik

Implementing `remove`

- Again, we need to shift elements in the array
 - this time, it's a left-shift
 - in what order should we process the elements?
 - what indexes should we process?

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	7	5	12	0	0	0	0
<i>size</i>	6									

- `list.remove(2);` *// delete 9 from index 2*

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	7	5	12	0	0	0	0	0
<i>size</i>	5 									

Implementing `remove` code

```
public void remove(int index) {  
    for (int i = index; i < size; i++) {  
        list[i] = list[i + 1];  
    }  
    size--;  
    list[size] = 0;        // optional (why?)  
}
```

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	7	5	12	0	0	0	0
<i>size</i>	6									

- `list.remove(2);` // delete 9 from index 2

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	7	5	12	0	0	0	0	0
<i>size</i>	5 ←									

Preconditions

- **precondition:** Something your method *assumes is true* at the start of its execution.

- Often documented as a comment on the method's header:

```
// Returns the element at the given index.  
// Precondition: 0 <= index < size  
public int get(int index) {  
    return elementData[index];  
}
```

- Stating a precondition doesn't really "solve" the problem, but it at least documents our decision and warns the client what not to do.
- What if we want to actually enforce the precondition?

Bad precondition test

- What is wrong with the following way to handle violations?

```
// Returns the element at the given index.  
// Precondition: 0 <= index < size  
public int get(int index) {  
    if (index < 0 || index >= size) {  
        System.out.println("Bad index! " + index);  
        return -1;  
    }  
    return elementData[index];  
}
```

- returning -1 no better than returning 0 (could be legal value)
- `println` is not a very strong deterrent to the client (esp. GUI)

Throwing exceptions (4.4)

```
throw new ExceptionType ();
```

```
throw new ExceptionType ("message");
```

- Generates an exception that will crash the program, unless it has code to handle ("catch") the exception.
- Common exception types:
 - `ArithmeticException`, `ArrayIndexOutOfBoundsException`, `FileNotFoundException`, `IllegalArgumentException`, `IllegalStateException`, `IOException`, `NoSuchElementException`, `NullPointerException`, `RuntimeException`, `UnsupportedOperationException`
- Why would anyone ever *want* a program to crash?

Exception example

```
public int get(int index) {  
    if (index < 0 || index >= size) {  
        throw new ArrayIndexOutOfBoundsException(index);  
    }  
    return elementData[index];  
}
```

- Exercise: Modify the rest of `ArrayIntList` to state preconditions and throw exceptions as appropriate.

Not enough space

- What to do if client needs to add more than 10 elements?

<i>index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>
<i>value</i>	3	8	9	7	5	12	4	8	1	6
<i>size</i>	10									

- `list.add(15);` **// add an 11th element**
- Possible solution: Allow the client to construct the list with a larger initial capacity.

Multiple constructors

- Our list class has the following constructor:

```
public ArrayIntList() {  
    elementData = new int[10];  
    size = 0;  
}
```

- Let's add a new constructor that takes a capacity parameter:

```
public ArrayIntList(int capacity) {  
    elementData = new int[capacity];  
    size = 0;  
}
```

- The constructors are very similar. Can we avoid redundancy?

this keyword

- **this** : A reference to the *implicit parameter*
(the object on which a method/constructor is called)
- Syntax:
 - To refer to a field: `this . field`
 - To call a method: `this . method (parameters) ;`
 - To call a constructor
from another constructor: `this (parameters) ;`

Revised constructors

// Constructs a list with the given capacity.

```
public ArrayList(int capacity) {  
    elementData = new int[capacity];  
    size = 0;  
}
```

// Constructs a list with a default capacity of 10.

```
public ArrayList() {  
    this(10);    // calls (int) constructor  
}
```

Class constants

```
public static final type name = value;
```

- **class constant:** a global, unchangeable value in a class
 - used to store and give names to important values used in code
 - documents an important value; easier to find and change later
- classes will often store constants related to that type
 - `Math.PI`
 - `Integer.MAX_VALUE`, `Integer.MIN_VALUE`
 - `Color.GREEN`

```
// default array length for new ArrayIntLists
```

```
public static final int DEFAULT_CAPACITY = 10;
```

Running out of space

- What should we do if the client starts out with a small capacity, but then adds more than that many elements?

<i>index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>
<i>value</i>	3	8	9	7	5	12	4	8	1	6
<i>size</i>	10									

- `list.add(15);` **// add an 11th element**

<i>index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>	<i>11</i>	<i>12</i>	<i>13</i>	<i>14</i>	<i>15</i>	<i>16</i>	<i>17</i>	<i>18</i>	<i>19</i>	
<i>value</i>	3	8	9	7	5	12	4	8	1	6	15	0	0	0	0	0	0	0	0	0	
<i>size</i>	11																				

- Answer: **Resize the array** to one twice as large.

The Arrays class

- The `Arrays` class in `java.util` has many useful methods:

Method name	Description
<code>binarySearch(array, value)</code>	returns the index of the given value in a <i>sorted</i> array (or <code>< 0</code> if not found)
<code>binarySearch(array, minIndex, maxIndex, value)</code>	returns index of given value in a <i>sorted</i> array between indexes <i>min</i> / <i>max</i> - 1 (<code>< 0</code> if not found)
<code>copyOf(array, length)</code>	returns a new resized copy of an array
<code>equals(array1, array2)</code>	returns <code>true</code> if the two arrays contain same elements in the same order
<code>fill(array, value)</code>	sets every element to the given value
<code>sort(array)</code>	arranges the elements into sorted order
<code>toString(array)</code>	returns a string representing the array, such as <code>"[10, 30, -25, 17]"</code>

- Syntax: `Arrays.methodName(parameters)`

Problem: size vs. capacity

- What happens if the client tries to access an element that is past the size but within the capacity (bounds) of the array?
 - Example: `list.get(7)` ; on a list of size 5 (capacity 10)

<i>index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>
<i>value</i>	3	8	9	7	5	0	0	0	0	0
<i>size</i>	5									

- Currently the list allows this and returns 0.
 - Is this good or bad? What (if anything) should we do about it?

Private helper methods

```
private type name (type name, ..., type name) {  
    statement(s);  
}
```

- a **private method** can be seen/called only by its own class
 - your object can call the method on itself, but clients cannot call it
 - useful for "helper" methods that clients shouldn't directly touch

```
private void checkIndex(int index, int min, int max) {  
    if (index < min || index > max) {  
        throw new IndexOutOfBoundsException(index);  
    }  
}
```

Postconditions

- **postcondition:** Something your method *promises will be true* at the end of its execution.
 - Often documented as a comment on the method's header:

```
// Makes sure that this list's internal array is large
// enough to store the given number of elements.
// Postcondition: elementData.length >= capacity
public void ensureCapacity(int capacity) {
    // double in size until large enough
    while (capacity > elementData.length) {
        elementData = Arrays.copyOf(elementData,
                                    2 * elementData.length);
    }
}
```

- If your method states a postcondition, clients should be able to rely on that statement being true after they call the method.