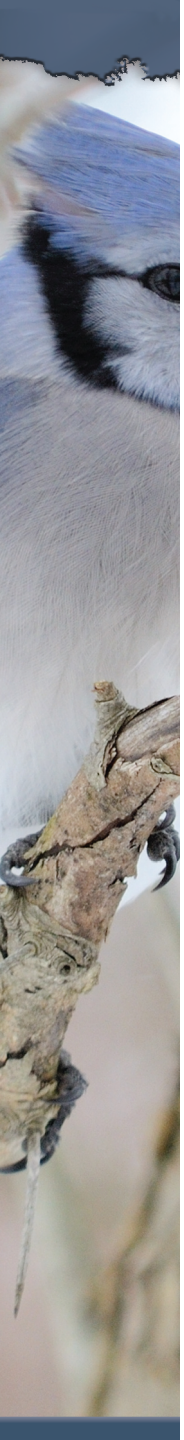
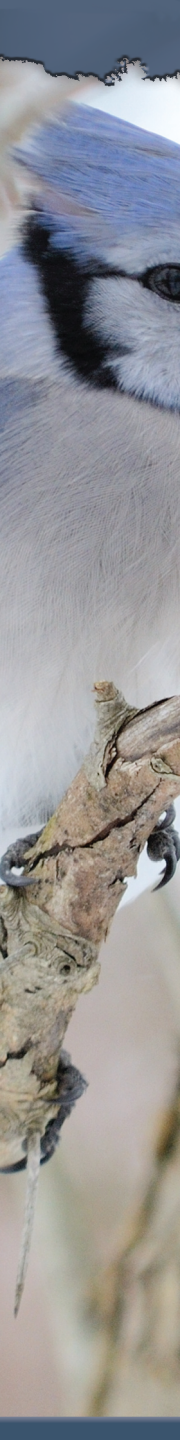


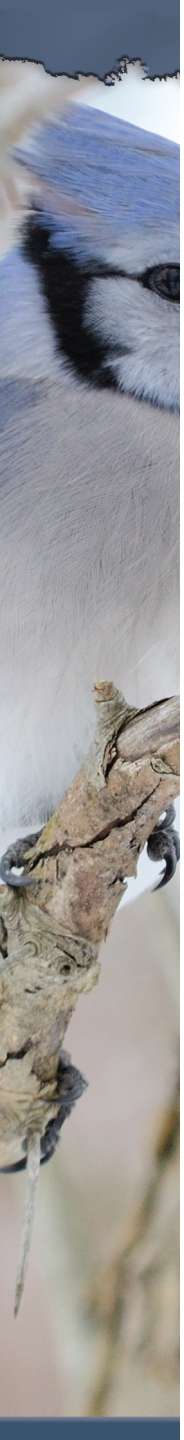
Fixed-sized collections

Introduction to arrays



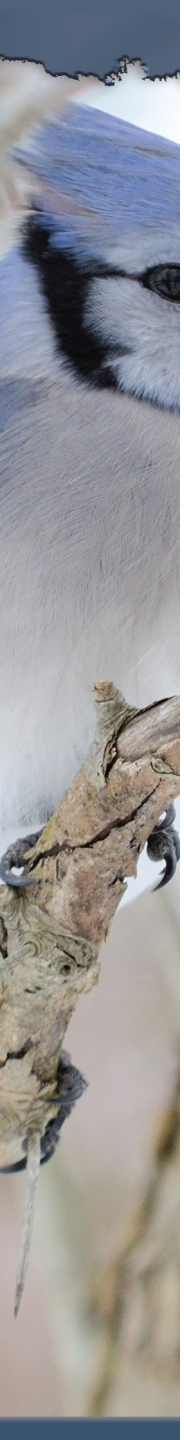
Features of arrays

- Fixed in length, unlike **`ArrayList`**, **`HashSet`**, **`HashMap`**, etc.
- Use a special syntax.
  - For historical reasons.
- Objects with no methods.
  - Methods are provided by other classes; e.g., **`java.util.Arrays`**.
  - Methods that are static.



Fixed-size collections

- Sometimes the maximum collection size can be pre-determined.
- A special fixed-size collection type is available: an *array*.
- Unlike the flexible List collections, arrays can store object references or primitive-type values (without autoboxing).



The *weblog-analyzer* project

- Web server records details of each access.
- Supports analysis tasks:
  - Most popular pages.
  - Busiest periods.
  - How much data is being delivered.
  - Broken references.
- Analyze accesses by hour – there is a fixed number of these in a day!

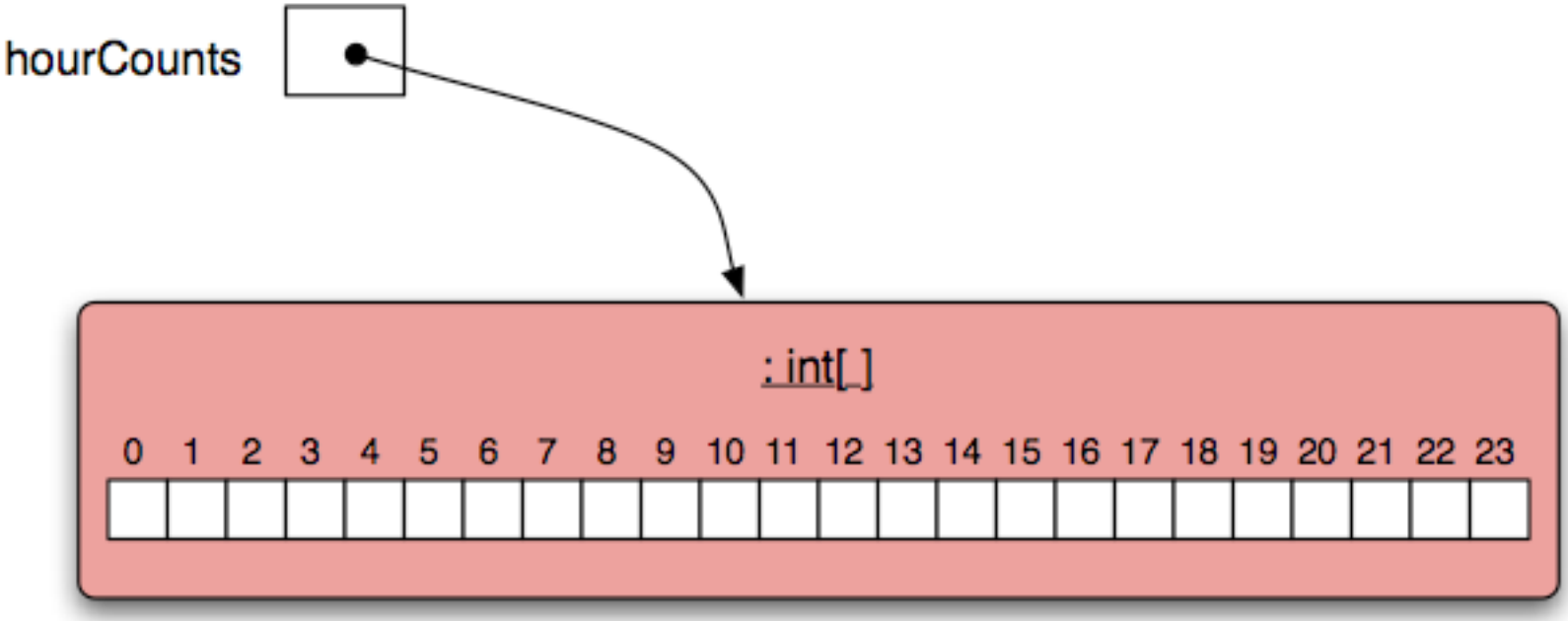
Creating an array object

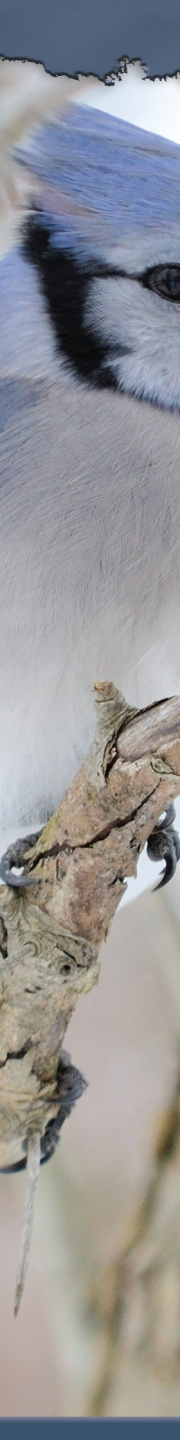
```
public class LogAnalyzer {
    private int[] hourCounts;
    private LogfileReader reader;

    public LogAnalyzer() {
        hourCounts = new int[24];
        reader = new LogfileReader();
    }
    ...
}
```

Array type
– does not contain size

Array object creation
– specifies size

The hourCounts array





Using an array

- Square-bracket notation is used to access an array element: `hourCounts[...]`
- Elements are used like ordinary variables.
- The target of an assignment:

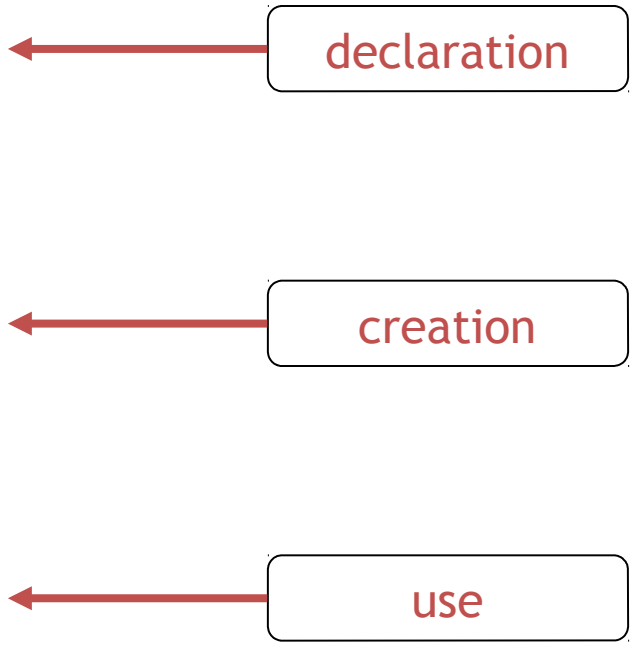
```
hourCounts[hour] = ...;
```

- In an expression:

```
hourCounts[hour]++;
if(hourCounts[hour] > 0) ...
```

Standard array use

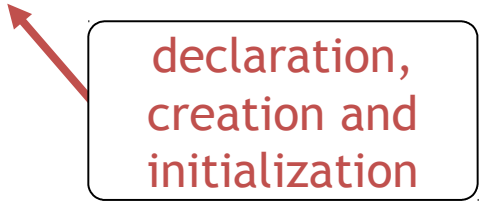
```
private int[] hourCounts;
private String[] names;

...

hourCounts = new int[24];

...

hourcounts[i] = 0;
hourcounts[i]++;
System.out.println(hourcounts[i]);
```



Array literals

- The size is inferred from the data.

```
private int[] numbers = { 3, 15, 4, 5 };
```



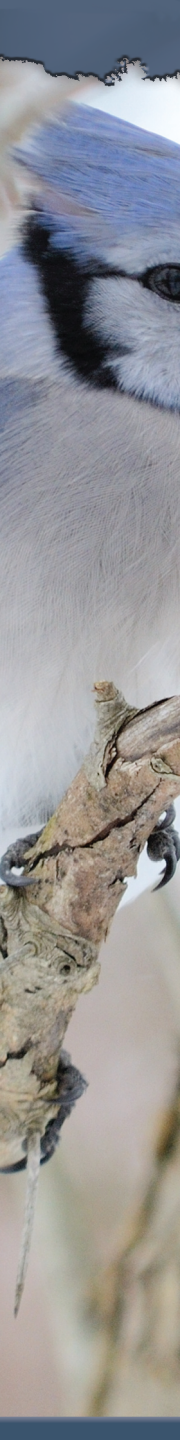
- Array literals in this form can only be used in declarations.
- Related uses require new:

```
numbers = new int[] {
    3, 15, 4, 5
};
```

Array length

```
private int[] numbers = { 3, 15, 4, 5 };

int n = numbers.length;
```

- NB: length is a field rather than a method.
- It's value cannot be changed – 'fixed size'.



The for loop

- There are two variations of the for loop, *for-each* and *for*.
- The for loop is often used to iterate a fixed number of times.
- Often used with a variable that changes a fixed amount on each iteration.

For loop pseudo-code

General form of the for loop

```
for(initialization; condition; post-body action) {
    statements to be repeated
}
```

Equivalent while-loop version

```
initialization;
while(condition) {
    statements to be repeated
    post-body action
}
```

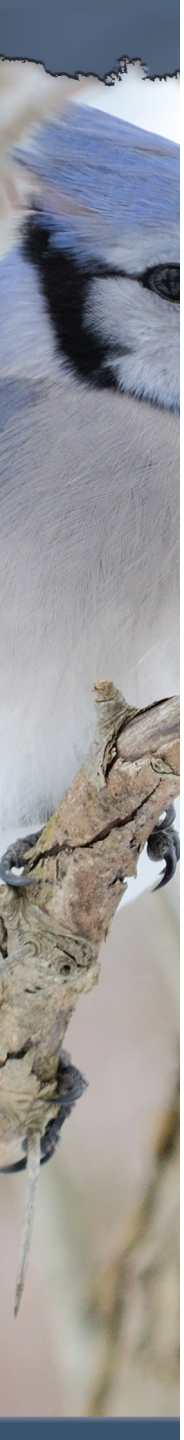
Array iteration

for loop version

```
for(int hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

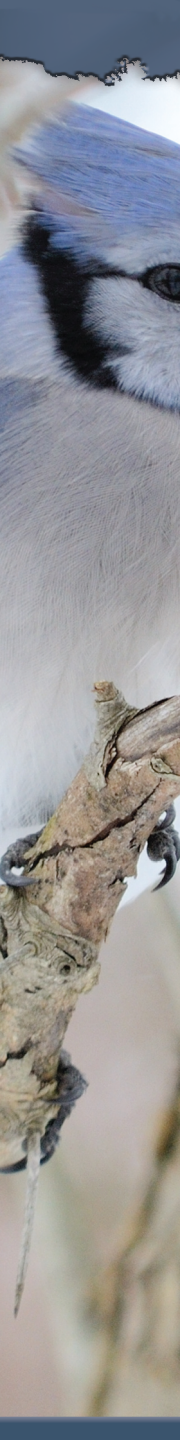
while loop version

```
int hour = 0;
while(hour < hourCounts.length) {
    System.out.println(hour + ": " + hourCounts[hour]);
    hour++;
}
```



Array-related methods

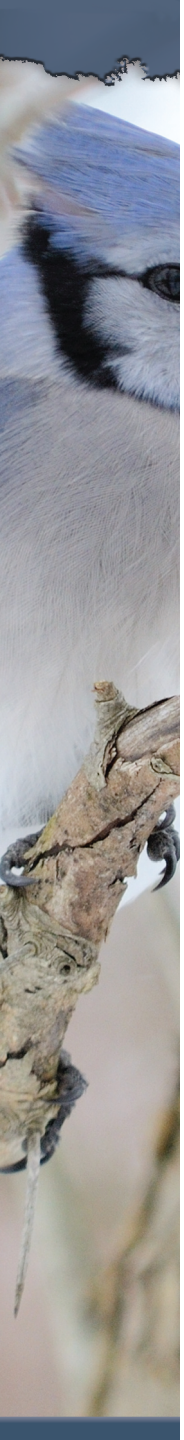
- **`System`** has static **`arraycopy`**.
- **`java.util.Arrays`** contains static utility methods for processing arrays:
  - **`binarySearch`**
  - **`fill`**
  - **`sort`**
- **`ArrayList`** has **`toArray`**.



Practice

- Given an array of numbers, print out all the numbers in the array, using a for loop.

```
int[] numbers = { 4, 1, 22, 9, 14, 3, 9};

for ...
```

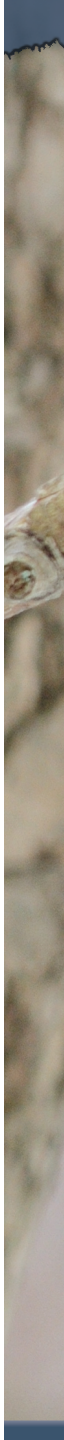


Practice

- Fill an array with the Fibonacci sequence.

```
0 1 1 2 3 5 8 13 21 34 ...
```

```
int[] fib = new int[howMany];

fib[0] = 0;
fib[1] = 1;

for(...) ...
```

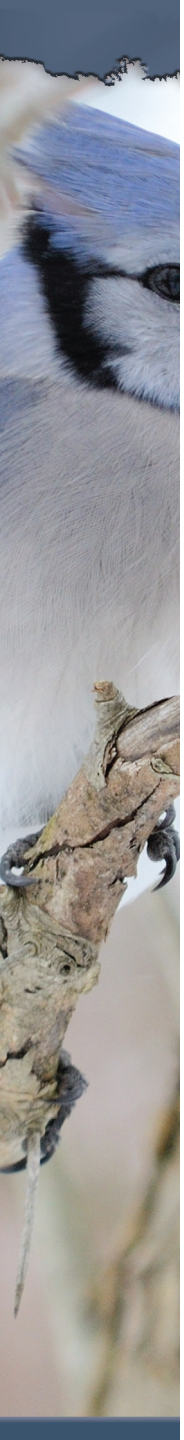
for loop with bigger step

```
// Print multiples of 3 that are below 40.
for(int num = 3; num < 40; num = num + 3) {
    System.out.println(num);
}
```

for loop and `Iterator`

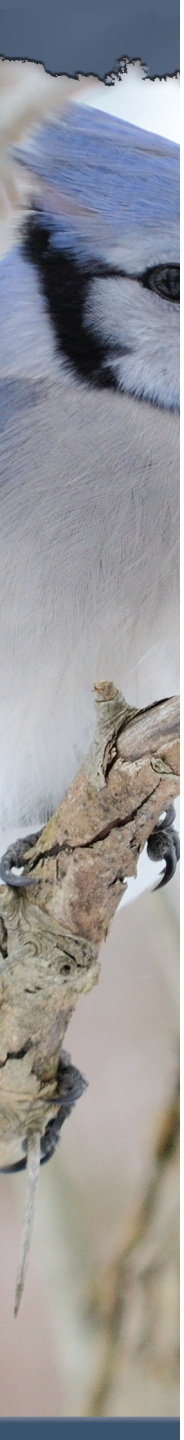
No post-body action required.

```
for(Iterator<Track> it = tracks.iterator();
                                    it.hasNext(); ) {
    Track track = it.next();
    if(track.getArtist().equals(artist)) {
        it.remove();
    }
}
```



Review

- Arrays are appropriate where a fixed-size collection is required.
- Arrays use a special syntax.
- For loops are used when an index variable is required.
- For loops offer an alternative to while loops when the number of repetitions is known.
- Used with a regular step size.



The *automaton* project

- An array of ‘cells’.
- Each cell maintains a simple state.
  - Usually a small numerical value.
  - E.g., on/off or alive/dead.
- The states change according to simple rules.
- Changes affected by neighboring states.

A simple automaton

```
nextState[i] =
     (state[i-1] + state[i] + state[i+1]) % 2;
```

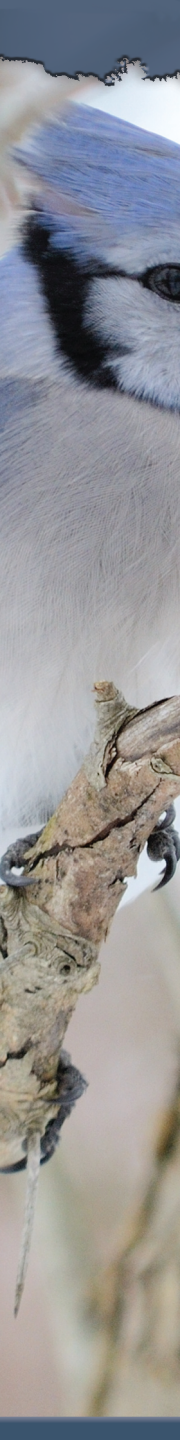
Step	Cell states - blank cells are in state 0												
0							+						
1						+	+	+					
2					+		+		+				
3				+	+		+		+	+			
4			+				+				+		
5		+	+	+		+	+	+		+	+	+	
6	+		+				+				+		+

The conditional operator

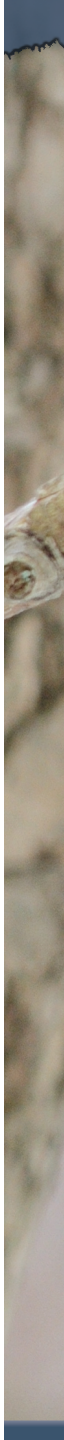
- Choose between two values:

```
condition ? value1 : value1
```

```
for(int cellValue : state) {
    System.out.print(cellValue == 1 ? '+' : ' ');
}
System.out.println();
```



Further advanced material



Arrays of more than one dimension

- Array syntax supports multiple dimensions.
  - E.g., 2D array to represent a game board, or a grid of cells.
- Can be thought of as an array of arrays.

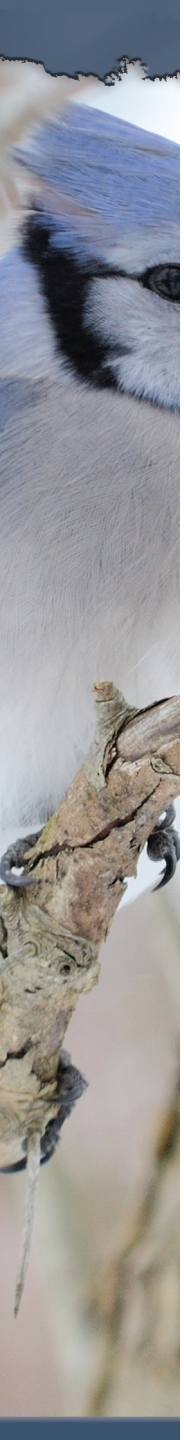
The *brain* project

```
Cell[][] cells;
...
cells = new Cell[numRows][numCols];
...
for(int row = 0; row < numRows; row++) {
    for(int col = 0; col < numCols; col++) {
        cells[row][col] = new Cell();
    }
}
```

Alternative iteration

```
for(int row = 0; row < cells.length; row++) {
    Cell[] nextRow = cells[row];
    for(int col = 0; col < nextRow.length; col++) {
        nextRow[col] = new Cell();
    }
}
```

- 'Array of array' style.
- Requires no access to **`numRows`** and **`numCols`**.
- Works with irregular shape arrays, which are supported in Java.



Arrays and Streams

- **`java.util.Arrays`** has several **`stream()`** methods that return a **`Stream`** based on an array.
- **`IntStream`** is a **`Stream`** of **`int`** values.
- **`Instream.range()`** creates a sequential **`IntStream`**.
- **`toArray`** returns an array from a **`Stream`**.