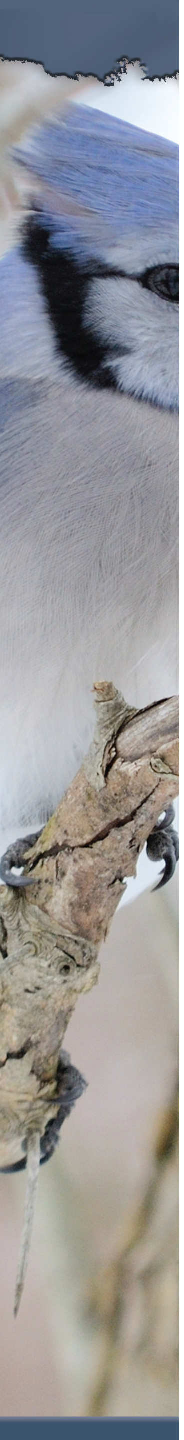
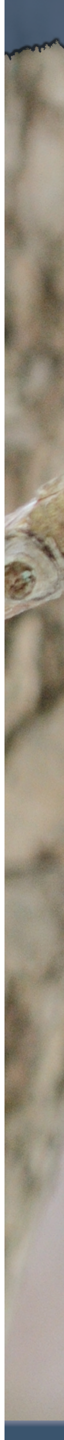
Fixed-sized collections

Introduction to arrays



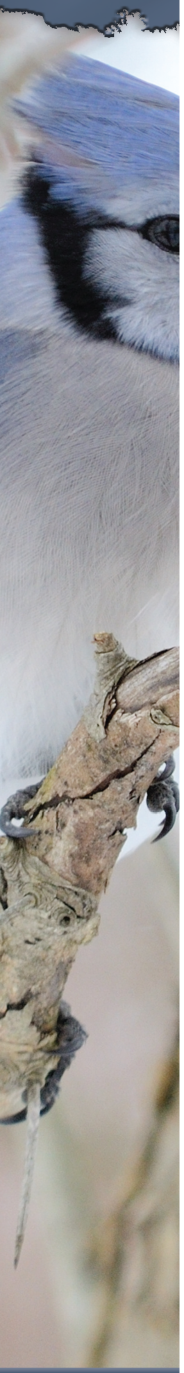
Features of arrays

- Fixed in length, unlike collections, eg, **`ArrayList`**.
- Use a special syntax.
  - For historical reasons.
- Objects with no methods.
  - Methods are provided by other classes; eg, **`java.util.Arrays`**.
  - Methods that are static (seen later).



Fixed-size collections

- Sometimes the maximum collection size can be pre-determined.
- A special fixed-size collection type is available: an *array*.
- Unlike collections, arrays can store:
  - object references
  - primitive-type values



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

Array type declaration
– does not contain size

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

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

Array object creation
– specifies size

(A word about final)

```
final type v = something;
```

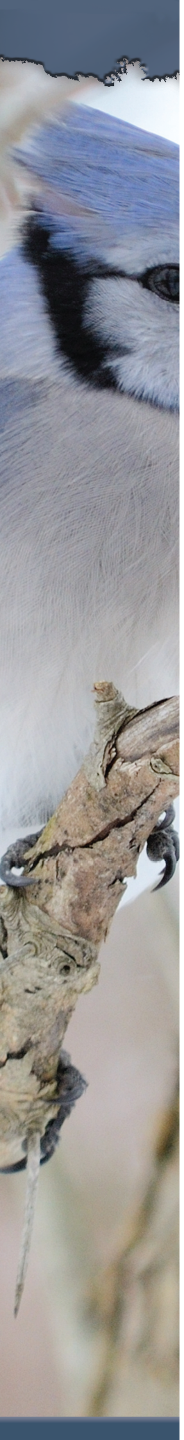
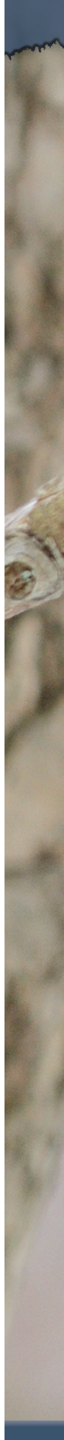
v : $\boxed{\text{something}}$

(A word about final)

```
final type v = something;

                    v : something


              v = somethingElse;
```

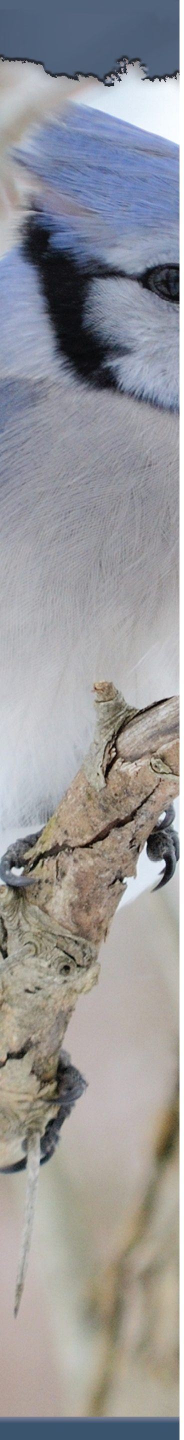
(A word about final)

```
final type v = something;
```

v : $\boxed{\text{something}}$

```
v = somethingElse;
```

Compile-time error!



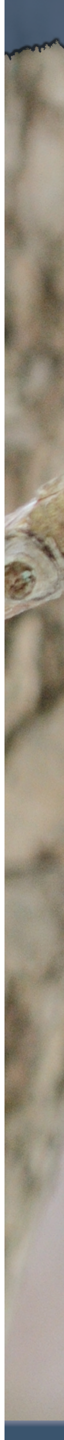
(A word about final)

```
final type v = something;
```

```
v : something
```

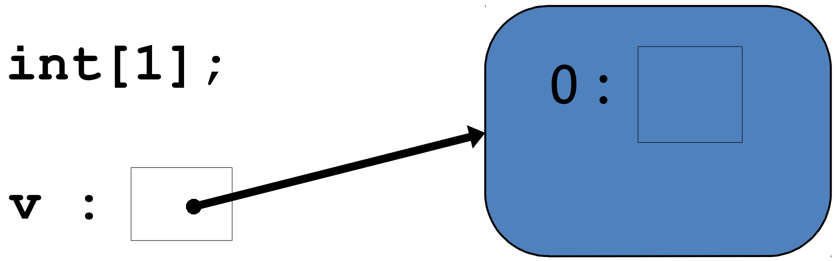
```
v = somethingElse;
```

Works exactly the same for primitive-type and reference-type



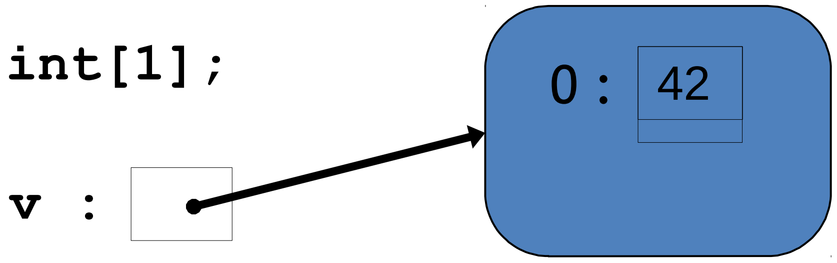
(But - final is **not** immutable)

```
final int[] v = new int[1];
```



(But - final is **not** immutable)

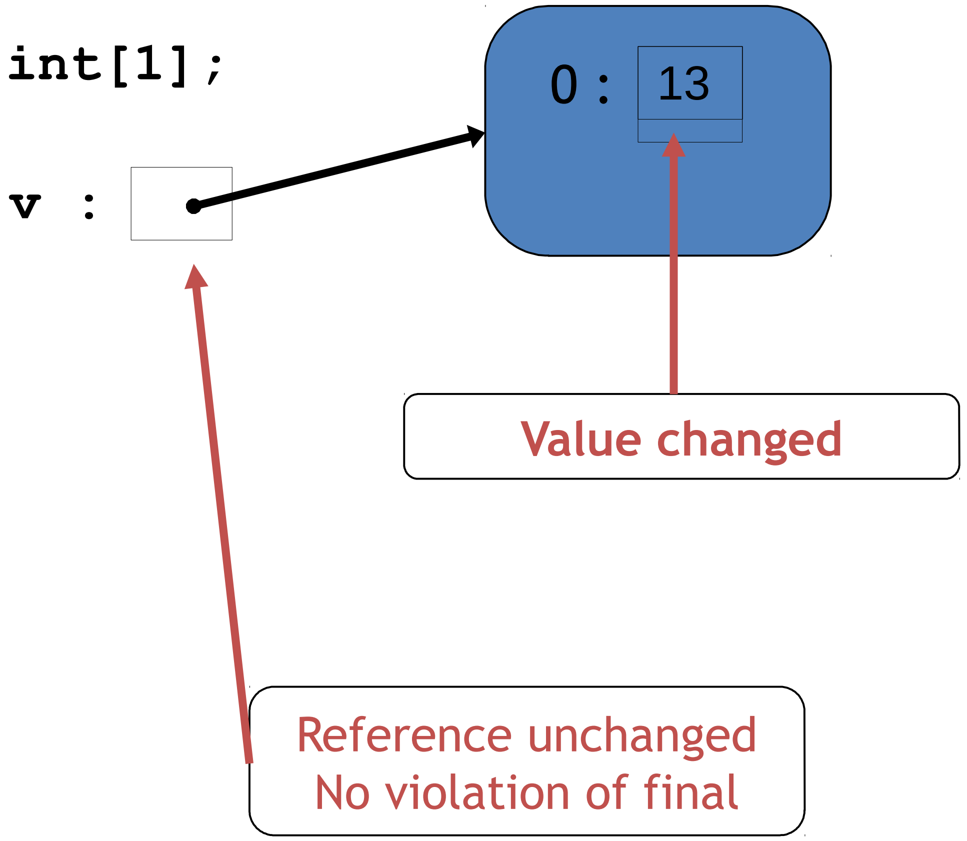
```
final int[] v = new int[1];
          v[0] = 42;
```



(But - final is **not** immutable)

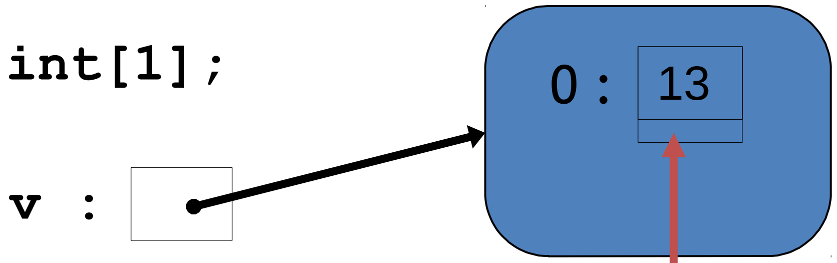
```
final int[] v = new int[1];
          v[0] = 42;

          v[0] = 13;
```



(But - final is **not** immutable)

```
final int[] v = new int[1];
          v[0] = 42;
```



```
          v[0] = 13;


             v = new int[1];
```

Value changed

Reference unchanged
No violation of final

Compile-time error!

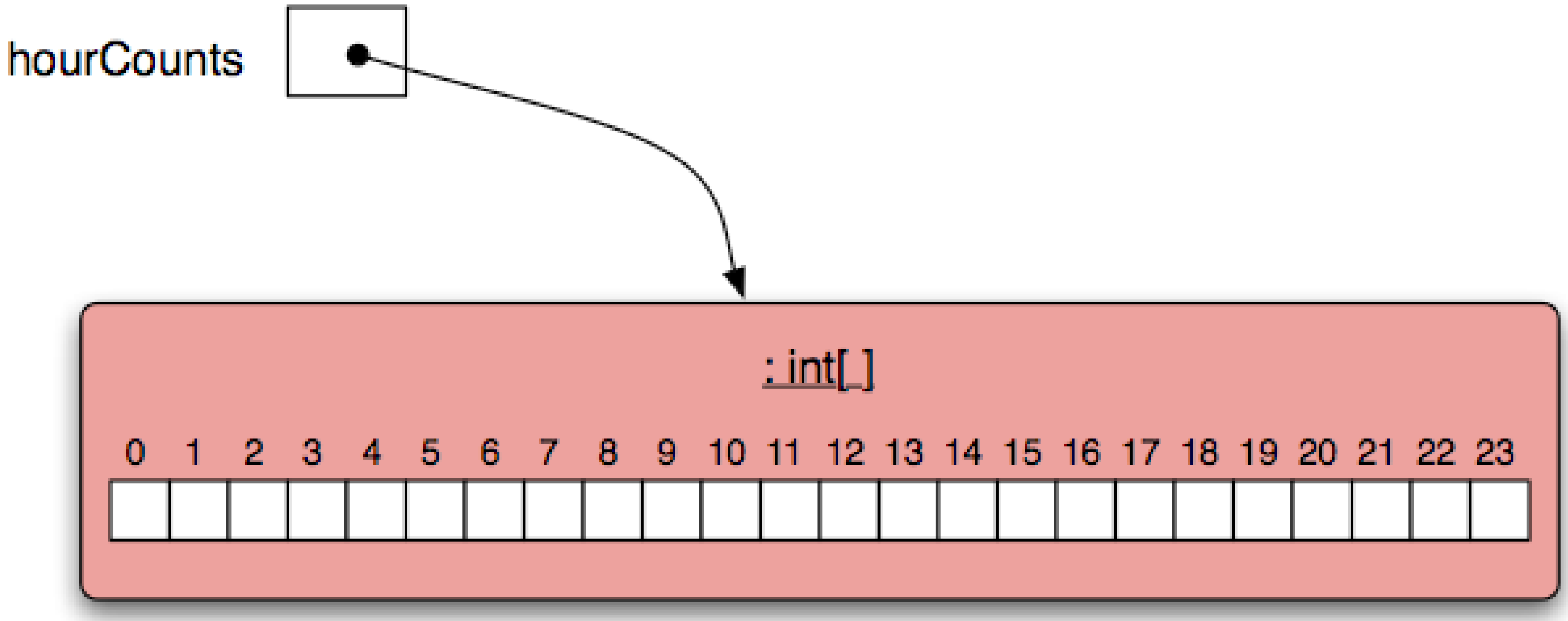
(`String` immutable by design)

```
final String baseName = "widget";
String partNum;
for (int i = 0; i < 1000; i++) {
    partNum = baseName + i;
    does something awesome with partNum
}
```

Creates a new String object ->
1000 objects

- So are **`Integer`**, **`Double`**, **`Boolean`**, etc.

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

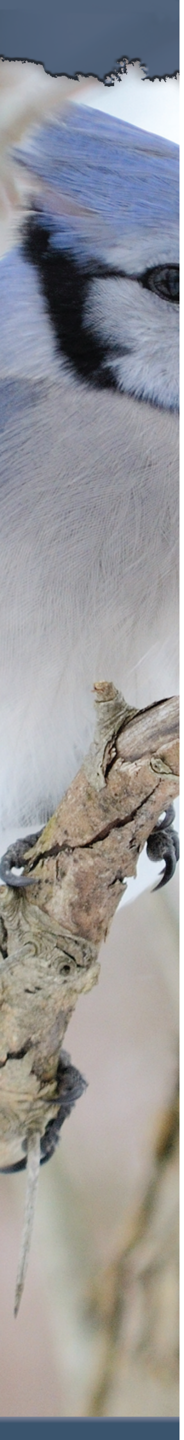
```
private final int[] hourCounts;
private final String[] names;

...

hourCounts = new int[24];

...

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



Standard array use

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

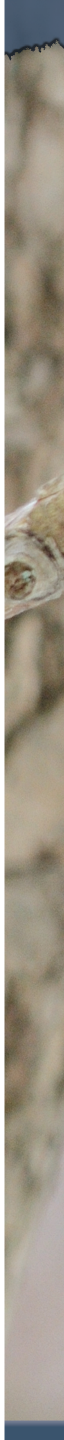
declaration

```
...

hourCounts = new int[24];

...

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



Standard array use

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

declaration

```
...

hourCounts = new int[24];
```

creation

```
...

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

Standard array use

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

...

hourCounts = new int[24];

...

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

declaration

creation

use

Array literals

- The size is inferred from the data.

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

declaration, creation and initialization

- 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 final int[] numbers = {3, 15, 4, 5};

int n = numbers.length;
```

not a method call!

- 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

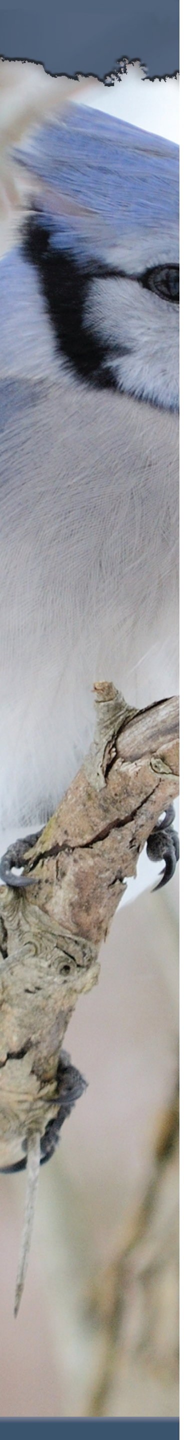
```
for (int hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

while loop version

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

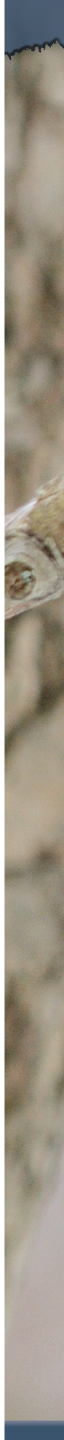
Array-related methods

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



Array-related methods

- **`List List.of(`***array***`)`**
  - Returns an **unmodifiable** list containing an arbitrary number of elements.
- **`List Arrays.asList(`***array***`)`**
  - Returns a fixed-size list backed by the specified array.
- *array arrayListObj***`.toArray()`**
  - Returns an array containing all of the elements in this list in proper sequence



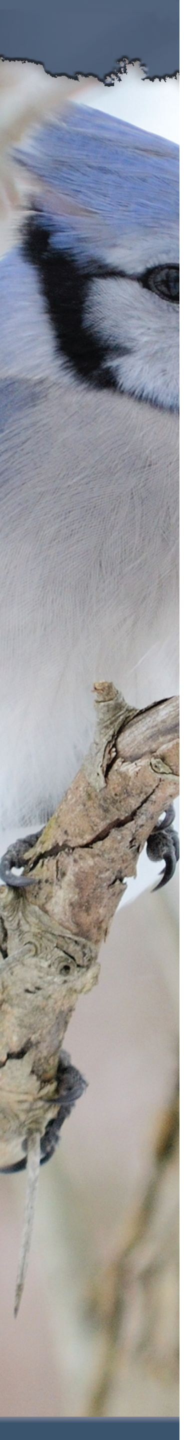
for loop with bigger step

```
// 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`

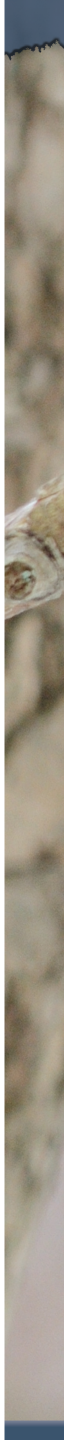
No post-body action required.

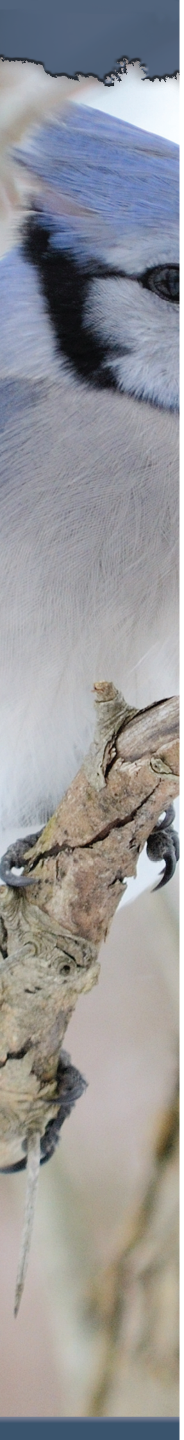
```
for (Iterator<Track> it = tracks.iterator();
            it.hasNext(); ) {
    Track track = it.next();
    if (track.getArtist().equals(artist)) {
        it.remove();
    }
}
```



Review

- Arrays are appropriate where a fixed-size collection is required.
- Arrays use a special syntax.
- Arrays have no methods.
- 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 conditional operator

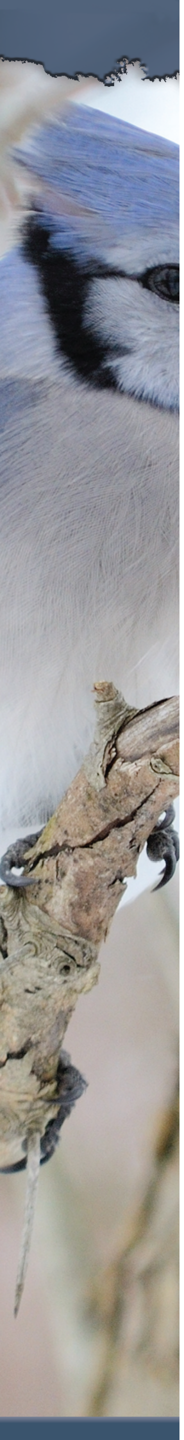
- Choose between two values:

```
condition ? value1 : value1
```

```
NumberDisplay hour;
…
System.out.print(
         (hour.value < 10 ? "0" : "")
        + hour.value
    );
```

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();
    }
}
```