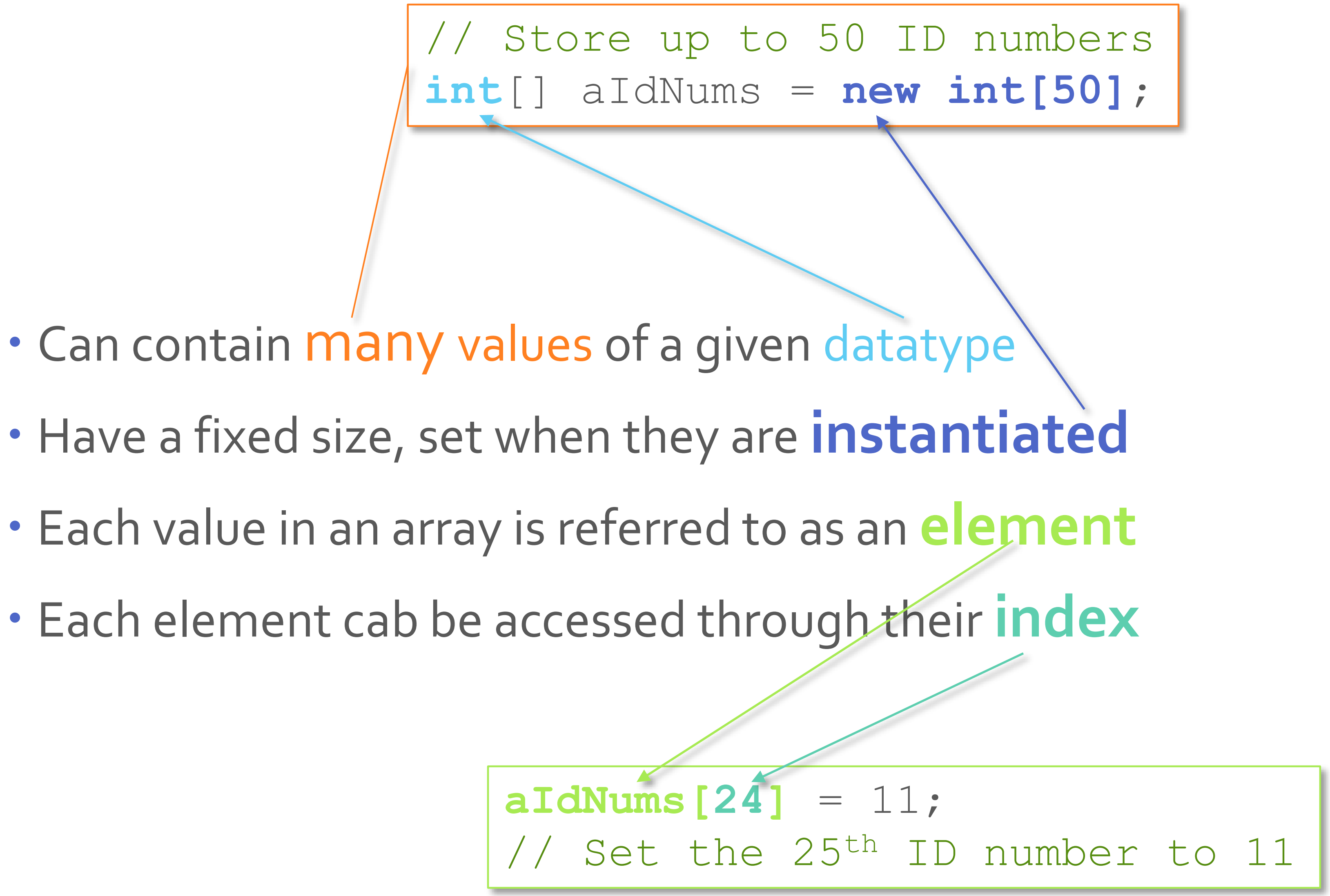
Lecture 2
Intro to Array Operations

Some core operations with Arrays.

Review

We got to know a new friend… the Array.

A crash course review of **ARRAYS**



```
// Store up to 50 ID numbers
int[] aIdNums = new int[50];
```

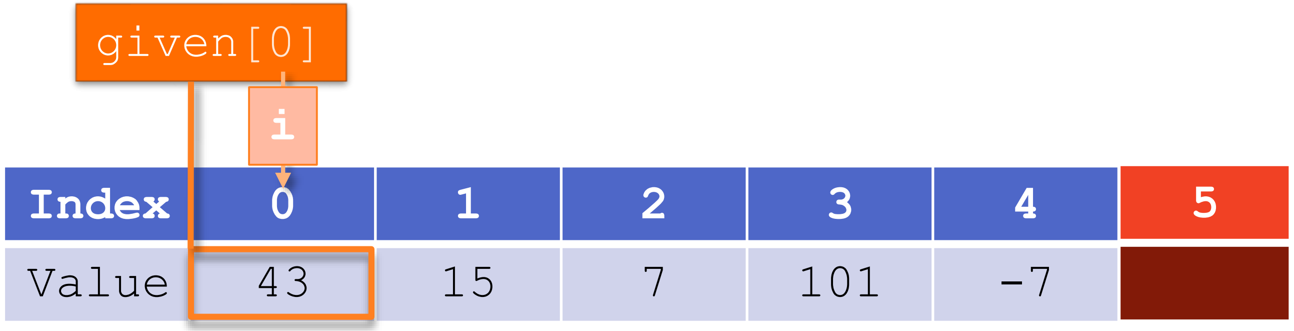
- Can contain many values of a given datatype
- Have a fixed size, set when they are **instantiated**
- Each value in an array is referred to as an **element**
- Each element cab be accessed through their **index**

```
aIdNums[24] = 11;
// Set the 25th ID number to 11
```

Operating on all of the elements with loops

```
public static void printIntArray(int[] given){
        for(int i = 0; i < given.length; i++){
                System.out.println(given[i]);
        }
}
```

- Suppose we are given an array with 5 elements (length = 5)
- We would need to access elements with index 0, 1, 2, 3, and 4
- The for-loop helps since it can count from 0 up to before the array length
- We can use the counter as a moving index to access every element



Index	0	1	2	3	4	5
Value	43	15	7	101	-7	

Last Meeting's Homework

Were you able to finish it? Where did you stop?

Problem

- **Ask the user for 5 ints.**
- Display the sum of the numbers.
- Display the numbers from highest to lowest.

We can use loops to **user-initialize** array elements

```
public static void scanIntArray(int[] given){
        Scanner sc = new Scanner(System.in);

        for(int i = 0; i < given.length; i++){
                System.out.print(“Enter an int:”);
        /* Instead of printing out each element,
        we can assign them user input instead */
                given[i] = sc.nextInt();
        }
}
```

- What problems did you encounter while doing this?

Testing these methods in main…

```
public static void printIntArray(int[] given)…
public static void scanIntArray(int[] given)…

public static void main(){
        int[] bunchOfInts = new int[5];
// Arrays are Objects, so they must be instantiated.
        scanIntArray(bunchOfInts);
        printIntArray(bunchOfInts);
}
```

This code has an error. Can you spot it?

Clue: It's not the syntax.

Problem

Σ

- Ask the user for 5 ints.
- **Display the sum of the numbers.**
- Display the numbers from highest to lowest.

Get **Sum**

```
public static int sumIntArray(int[] given){
      int sum = 0;
      for(int i = 0; i < given.length; i++){
            sum += given[i];
      }
      return sum;
}
```

Checkpoint

✓Ask the user for 5 ints (with a loop)

✓Get and display the sum of the numbers

✓Test by running in main

✓Should be easy to extend from **5** ints to any number **n**.

Let’s a try a slightly different problem

- Display from highest to lowest.

- Let’s try just getting the **highest** value from an array

Exercise Problem

- Create a method named `getHighest` that accepts an int[] parameter. This method returns the largest value in the given array. The values can be negative or positive.

Get Highest

You should already be familiar with this problem!

```
public static int getHighest(int[] given){
      int high = given[0];

      for(int i = 1; i < given.length; i++){
         if(high < given[i])
            high = given[i];
      }


      return high;
}
```

Some adjustments

- Instead of the highest value, modify the method to return the **index** of the highest value in the array.

Issues

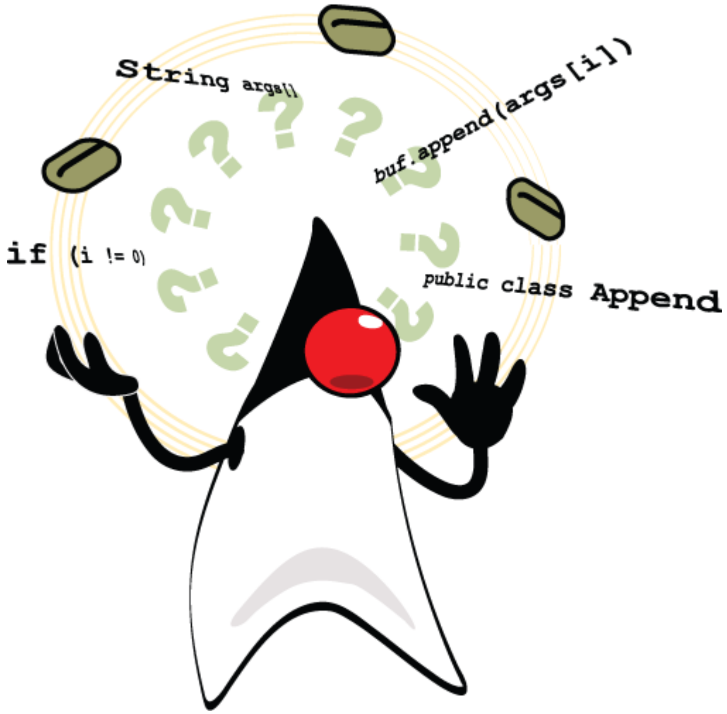
- We only got the highest but there are still 4 others (or even n-1 others)

- Out current approach does not recognize the order of the numbers, just the highest one.

Lecture 2 End

The remaining slides after the Exercise are a **Reading Assignment**

Reading Assignment: The Java API



Makes life easier if you know where to look.

But only makes it harder if you don’t.

Problem

- Ask the user for 5 ints.
- Display the sum of the numbers.
- **Display the numbers from highest to lowest.**

This time, let’s try putting the whole list in order first.

Sorting algorithm (Bubble sort)

```
public static void sortIntArrayDsc(int[] given){
        int temp;
        for (int i = 0; i < given.length; i++){
                for (int j = 1; j < given.length-i; j++){
                        if (given[j-1] < given[j]){
                                temp = given[j-1];
                                given[j-1] = given[j];
                                given[j] = temp;
                        }
                }
        }
}
```

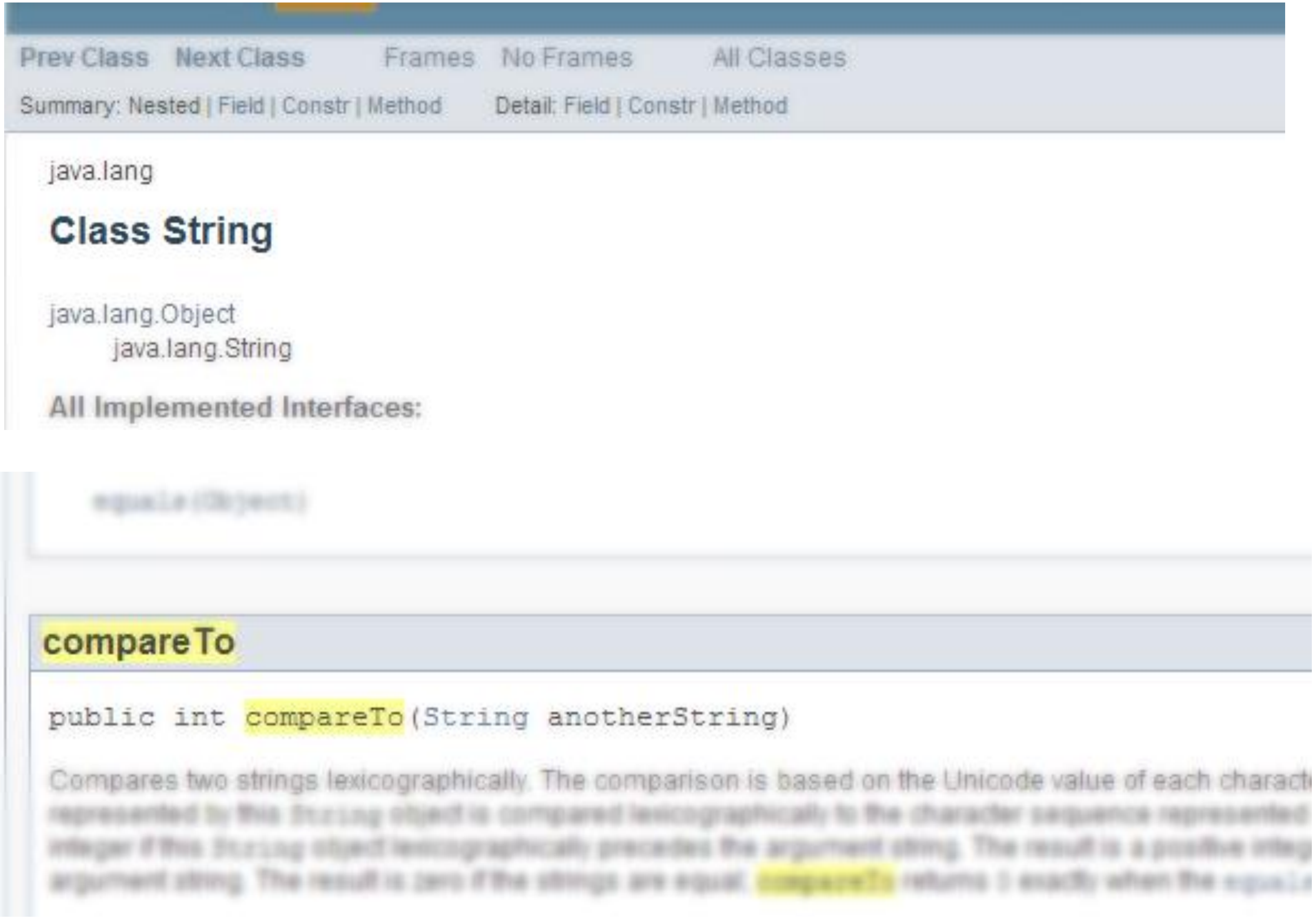
Search Google for explanations of the bubble sort algorithm.

The final solution

```
public static void printIntArrayDsc(int[] given){
      sortIntArrayDsc(given);
      printIntArray(given);
}
```

- Problems with this solution
  - Requires knowledge of sorting algorithms
  - Requires implementing the sorting algorithm
  - A similar solution is already available in Java

We’ve been here before (1st meeting)


Prev Class
Next Class
Frames
No Frames
All Classes
Summary: Nested | Field | Constr | Method
Detail: Field | Constr | Method

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

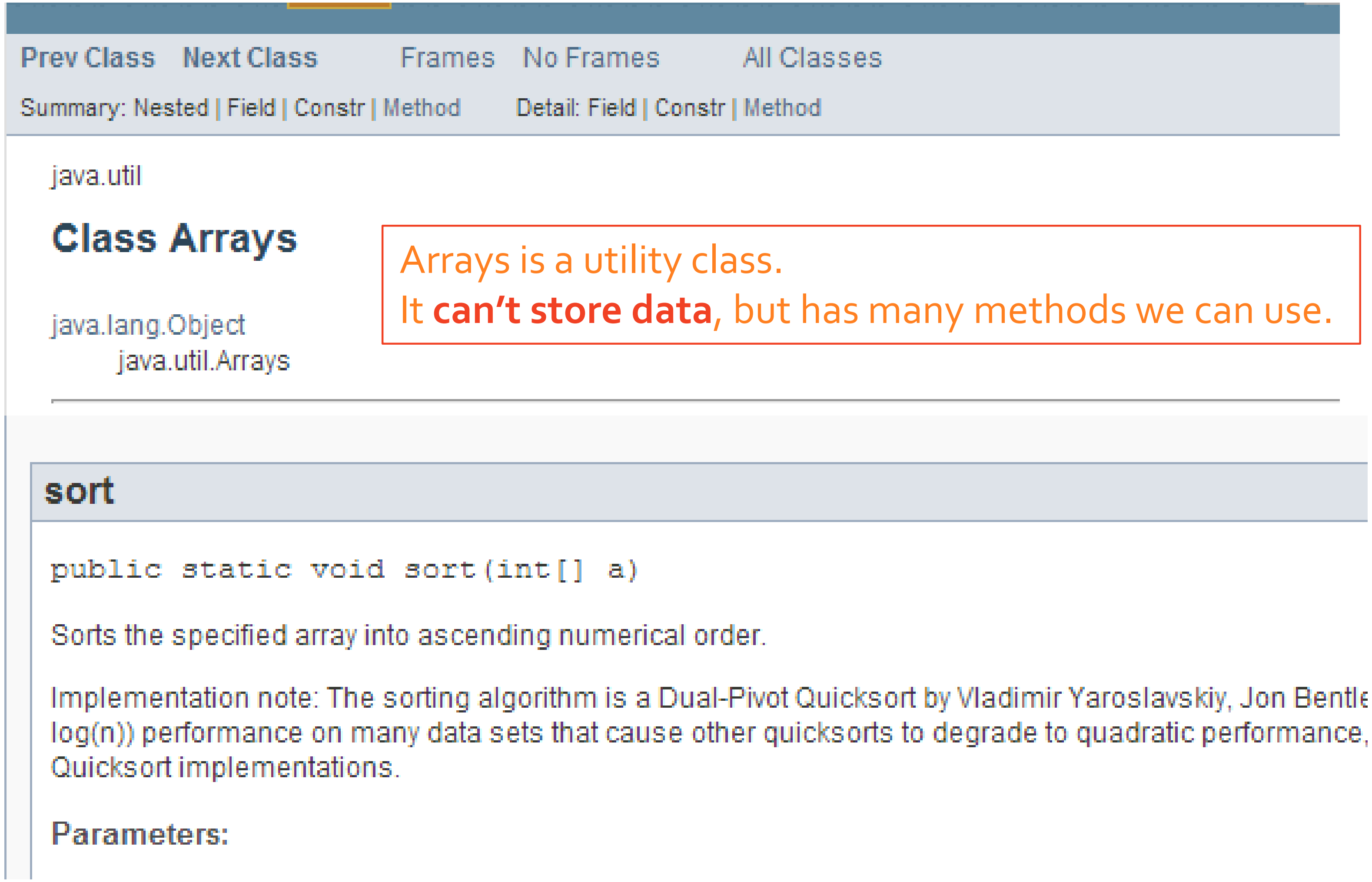
equals(Object)

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each charact
represented by this String object is compared lexicographically to the character sequence represented
integer if this String object lexicographically precedes the argument string. The result is a positive integ
argument string. The result is zero if the strings are equal; compareTo returns 0 exactly when the equals

Arrays is not Array(s)



Prev Class   Next Class   Frames   No Frames   All Classes
Summary: Nested | Field | Constr | Method   Detail: Field | Constr | Method

java.util

Class Arrays

java.lang.Object
    java.util.Arrays

Arrays is a utility class.
It **can't store data**, but has many methods we can use.

sort

```
public static void sort(int[] a)
```

Sorts the specified array into ascending numerical order.

Implementation note: The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentle log(n)) performance on many data sets that cause other quicksorts to degrade to quadratic performance, Quicksort implementations.

Parameters:

Can you figure out what the method above does?

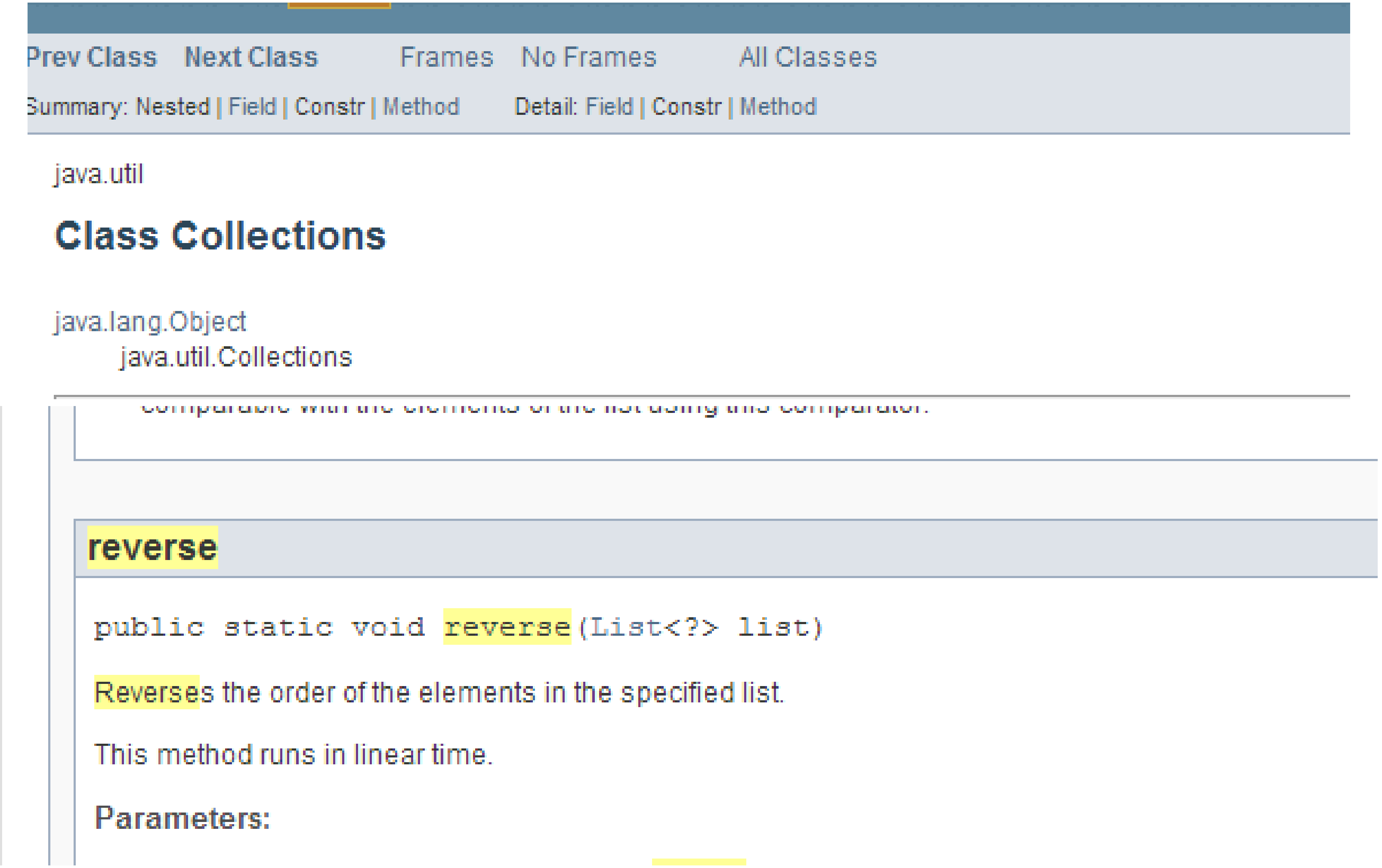
The alternative solution

```
import java.util.Arrays;


public static void printIntArrayDsc(int[] given){
      Arrays.sort(given);
      printIntArray(given);
}
```

- Problems with this solution
  - ~~Requires knowledge of sorting algorithms~~
  - ~~Requires implementing the sorting algorithm~~
  - ~~A similar solutions is already available in Java~~
  - **Contents are in the wrong order (ascending)**

Java to the rescue



Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class Collections

java.lang.Object
  java.util.Collections

comparable with the elements of the list using this comparator.

reverse

```
public static void reverse(List<?> list)
```

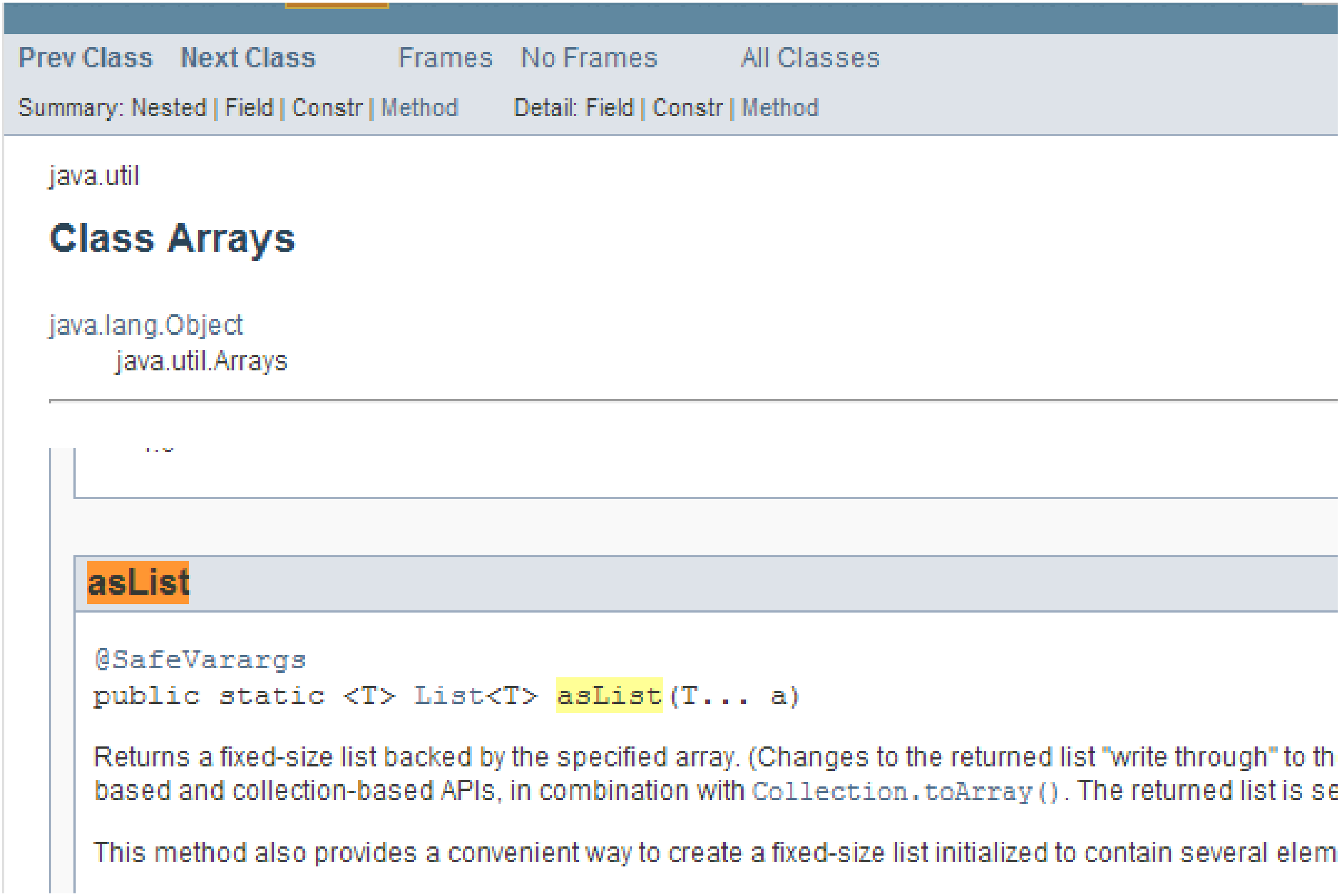
Reverses the order of the elements in the specified list.

This method runs in linear time.

Parameters:

- Buuut… this method needs a **List**, but we have an Array (wrong datatype).

Java to the rescue (again)



Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class Arrays

java.lang.Object
    java.util.Arrays

asList

```
@SafeVarargs
public static <T> List<T> asList(T... a)
```

Returns a fixed-size list backed by the specified array. (Changes to the returned list "write through" to th
based and collection-based APIs, in combination with `Collection.toArray()`. The returned list is se

This method also provides a convenient way to create a fixed-size list initialized to contain several elem

- T… basically means any type of array.

The final alternative solution

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public static void printIntList(List<Integer> given) {
        for (int i = 0; i < given.size(); i++) {
                System.out.println(given.get(i));
        }
}

public static void printIntArrayDsc(int[] given){
        Arrays.sort(given);
        List<Integer> temp = new ArrayList(Arrays.asList(arr));
        Collections.reverse(temp);
        printIntList(temp);
}
```

We needed to use all of these other Classes.

Notice that we needed to convert int[] to Integer[].
We had to because the built-in API methods required it.

Not always better

```
public static void printIntArrayDsc(int[] given){
        Arrays.sort(given);
        List<Integer> temp = new ArrayList(Arrays.asList(arr));
        Collections.reverse(temp);
        printIntList(temp);
}
```

- Problems with this solution
  - Requires knowledge of Class/Object usage
  - Requires extensive knowledge of the Java API
    - Not all operations are immediately available
  - Limited compatibility with primitive types
    - Have to convert int to Integer for the Java API

Summary

Programming Your Own

- + **Flexible** – easy to modify in case of changing requirements
- + **Transparent** – you can see and check how the code works (to update or debug)
- + **Customized** – methods cater specifically for the requirements of the organization
- ✕ **Heavily algorithmic** – need familiarity of any required algorithms
- ✕ **Longer code** – adds more code to your programming projects

Using Prebuilt API Methods

- + **Algorithmically simple** – you don't need to know algorithms
- + **(Usually) Efficient** – algorithms used are usually better than basic ones
- + **Shorter code** – only see method calls and high-level algorithms
- ✕ **API knowledge** – need familiarity of one or more API classes and functionalities
- ✕ **Rigid implementation** – if it's not supported, it can be tricky to get the functionality needed

The bottom-line

- You can use either approach for your exercises and projects.
- Choose wisely so you don’t waste time with trial and error or forcing things to work.
- Using a balanced combination of both is usually best.

~ End ~

- Go and try out the new stuff you’ve learned!