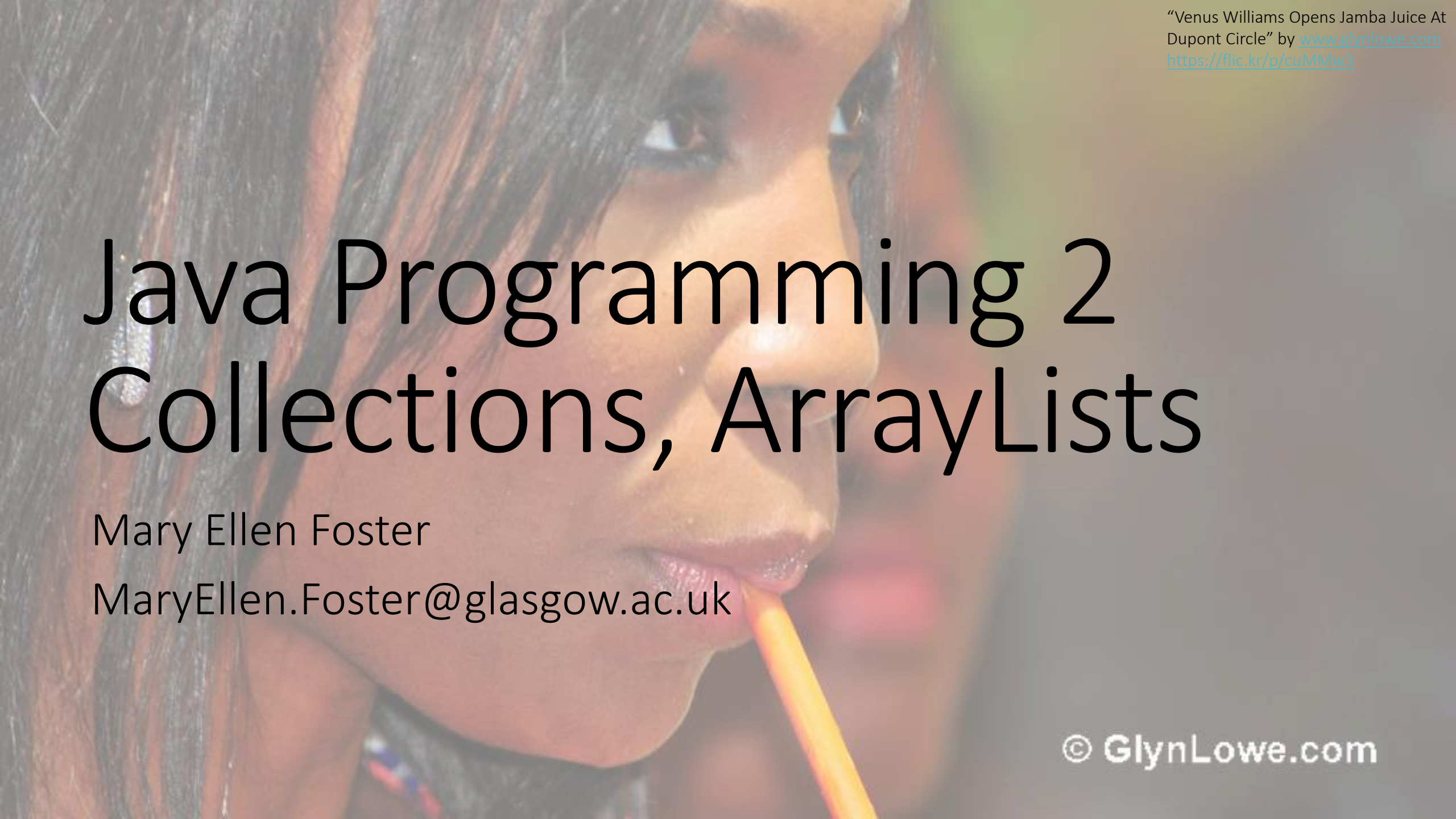


"Venus Williams Opens Jamba Juice At Dupont Circle" by www.glynlowe.com https://flic.kr/p/cuMMw3

Java Programming 2 Collections, ArrayLists

Mary Ellen Foster

MaryEllen.Foster@glasgow.ac.uk

Arrays revisited

Why use an array?

- Storing a group of values
- Directly supported by underlying Java Virtual Machine (JVM) – **efficient**

Major limitation: **fixed size**

- Size is determined when array is created

```
int[] numbers = new int[10];
String[] strings = { "each", "peach", "pear", "plum" };
```

- If you go past the end, you get an `ArrayIndexOutOfBoundsException`
- If you don't use all the space, you have wasted the additional capacity

Java Collections framework

A standard set of built-in classes for representing and manipulating collections

Each Collections class groups **related elements** into a **single unit**

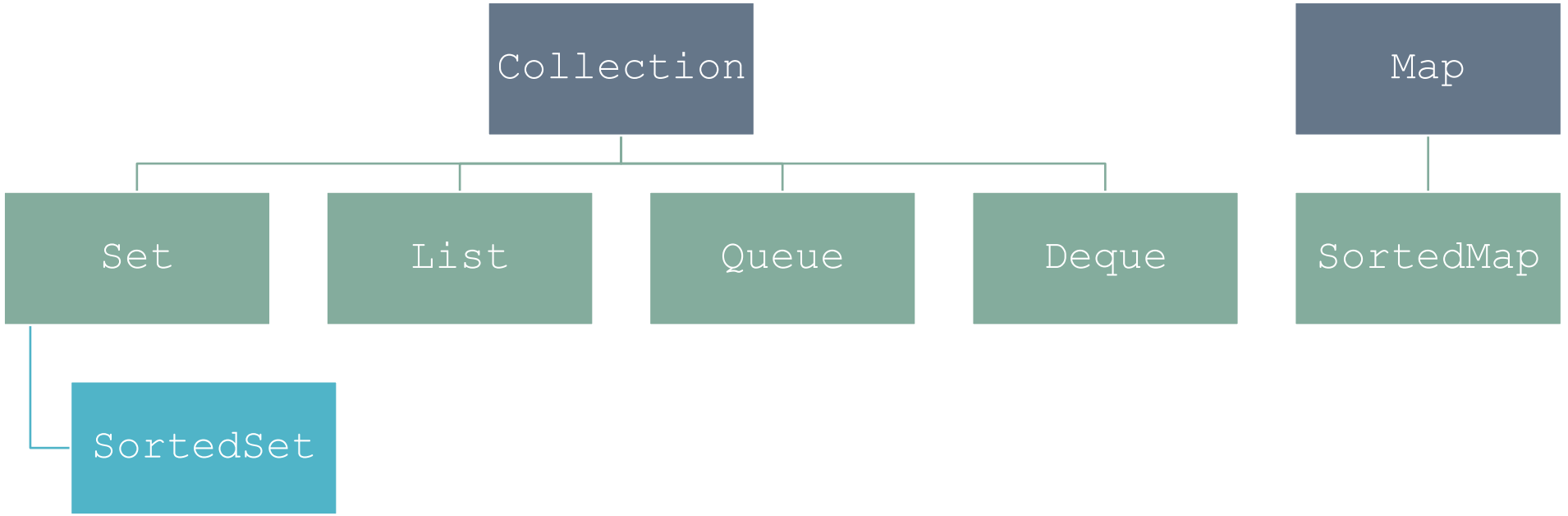
Examples:

- **ArrayList** – acts like a variable-length array
- **HashSet** – a group of unique elements
- **Stack** – a list with last-in/first-out semantics
- **HashMap** – a **dictionary** (e.g., a telephone directory)

Structure of Collections framework

Base class:* `java.util.Collection`

Methods: `add(), remove(), contains(), size(), toArray()`



** Actually, everything in this picture is an interface*

Advantages of Collections

Reduces programming effort by providing pre-written data structures and algorithms

Increases performance by providing high-performance implementations
- Implementations are interchangeable – can switch to tune performance

Provides interoperability by allowing Collections to be passed back and forth

Reduces effort to learn new APIs by providing a common interface

Reduces effort to design APIs by giving design specifications

Fosters software reuse by providing a standard interface

(List adapted from https://docs.oracle.com/javase/tutorial/collections/intro/index.html)

java.util.ArrayList

A Collections class (specifically, a `List`) that implements **variable-length** arrays

More flexible than built-in arrays, but less efficient

Acts as a wrapper around an underlying array that grows and shrinks dynamically

`ArrayList` is a **class** – so elements are added and removed by **methods**
(Not by built-in Java syntax as with normal arrays)

It has a **capacity** (size of internal array) and a **size** (number of elements in the list)

Capacity is increased when necessary – purely internal

Size is increased/decreased as elements are added and removed, and checked for operations

In general: IndexOutOfBoundsException if `(index < 0 || index >= size())`

Creating an ArrayList

```
// Default initial capacity (10)
List<String> strings = new ArrayList<>();

// Explicit initial capacity
List<String> strings = new ArrayList<>(50);
```

Size and capacity

```
// Returns size
int size = strings.size();

// Checks whether list is empty (i.e., is size == 0)
if (! strings.isEmpty() ) { … }

// Trims capacity to current size
strings.trimToSize();

// Ensures minimum capacity
strings.ensureCapacity(100);
```

Adding elements

```
// Adds the element to the end of the list
// Always succeeds; increases capacity and/or size as necessary
strings.add ("foo");


// Adds the element at the given index, and shifts other elements
// May throw IndexOutOfBoundsException
strings.add (5, "foo");


// Sets the element at the given index to the new value
// May throw IndexOutOfBoundsException
strings.set (5, "foo");
```

Accessing and removing elements

```
// Returns element at the given position
// May throw IndexOutOfBoundsException
String s = strings.get(5);


// Removes (and returns) element at the given position
// Shifts all remaining elements to the left
// May throw IndexOutOfBoundsException
String s = strings.remove(5);


// Removes first occurrence of given element in list
// Shifts all remaining elements to the left
// Returns true if element was there, and false if not
if (strings.remove ("foo") ) { … }
```

Checking list contents

```
// Returns true if the list contains the given element
if (s.contains (“foo”)) { … }


// Returns index of first occurrence of element in list
// (or -1 if it’s not there)
int i = strings.indexOf (“foo”);


// Returns index of last occurrence of element in list
// (or -1 if it’s not there)
int i = strings.lastIndexOf (“foo”);
```

Array vs ArrayList at a glance

Operation	Array	ArrayList
Declaration	`String[] strings;`	`ArrayList<String> strings;`
Initialisation	`strings = new String[10];`	`strings = new ArrayList<>(10);`
Setting element	`strings[5] = "foo";`	`strings.set(5, "foo");`
Accessing element	`String s = strings[5];`	`String s = strings.get(5);`
Getting size	`int n = strings.length;`	`int n = strings.size();`
Adding element	*n/a*	`strings.add("foo");` `strings.add(5, "foo");`
Removing element	*n/a*	`strings.remove("foo");` `strings.remove(5);`

Finding element	`Arrays.binarySearch(strings, "foo");`	`strings.contains("foo");`
		`strings.indexOf("foo");`
		`strings.lastIndexOf("foo");`

`List` vs `ArrayList`?

`List` is the **high-level Collection type** (actually it's an interface)
- Specifies methods including **add**, **clear**, **isEmpty**, **remove**, **set**, …

`ArrayList` is the **specific type of List**
- Provides a concrete implementation
- Additional methods related to capacity

When to use which?
- Use `ArrayList` …
  - *When initialising a new variable*
  - *If you want to manipulate capacity*
- Use `List` all other times – allows implementations to be swapped cleanly

Converting to and from normal arrays

```
// Convert a List to an array
List<String> strList = new ArrayList<>();
String[] strArray =
         strList.toArray(new String[strList.size()]);

// Convert an array to a List
String[] strings = { “each”, “peach”, “pear”, “plum” };
List<String> stringList = Arrays.asList (strings);
```

Iterating over ArrayLists – same as arrays

```
for (int i = 0; i < words.size(); i++) {
    String s = words.get (i);
    System.out.print (s + " ");
}
// or …
for (String s : words) {
    System.out.print (s + " ");
}
```

Bonus: ArrayList has `toString()`!

```
String[] words = { “each”, “peach”, “pear”, “plum” };
System.out.println (words);
// Prints “[Ljava.lang.String;@659e0bfd”

List<String> wordList = Arrays.asList (words);
System.out.println (wordList);
// Prints “[each, peach, pear, plum]”
```