More on Arrays

- Arrays of objects
- Command line arguments
- The `ArrayList` class
- Javadoc
- Review Lecture 8 notes and L&L 7.1 – 7.2
- Reading for this lecture: L&L 7.3 – 7.7, App I

Arrays of Objects

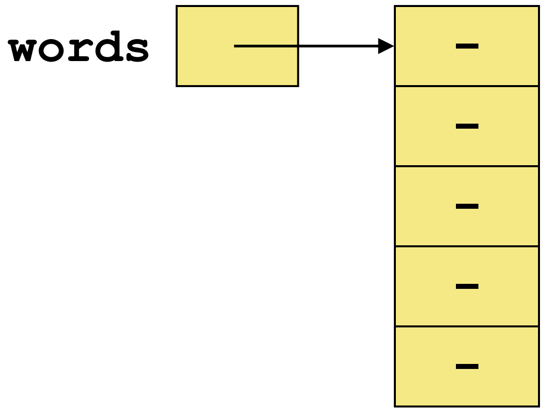
- The elements of an array can be object references
- The following declaration reserves space to store 5 references to `String` objects

```
String[] words = new String[5];
```

- It does NOT create the `String` objects themselves
- Initially an array of objects holds `null` references
- Each object stored in an element of an array must be instantiated separately

Arrays of Objects

- The `words` array when initially declared:



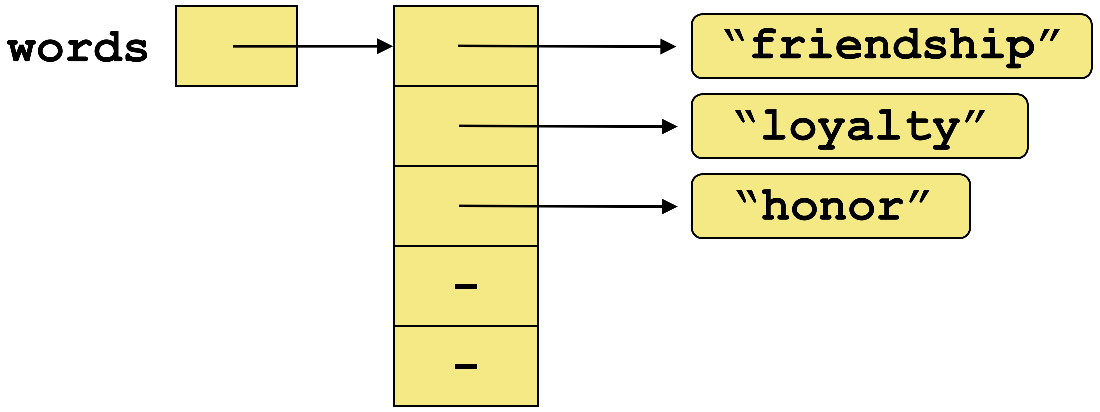
- A reference to `words.length` is OK (= 5)
- However, the following reference will throw a `NullPointerException`:

```
System.out.println(words[0].length());
```

Arrays of Objects

- To create some `String` objects and store them in elements of the array:

```
words[0] = new String("friendship");
words[1] = "loyalty";
words[2] = "honor";
```



Arrays of Objects

- `String` objects can be created using literals
- The following declaration creates an array object called `verbs` with a length of 4 and fills it with references to four `String` objects created using string literals

```
String[] verbs = {"play", "work", "eat", "sleep"};
```

Arrays of Objects

- To use one of the methods of an object element of an array:

```
verbs[2].equals(“eat”);  // true
```

- To pass one of the object elements of an array as a parameter to a method:

```
“eat”.equals(verbs[2]);  // true
```

- To return an element of an array:

```
public String methodName(String [] verbs)
{
  return verbs[2];          // “eat”
}
```

Command-Line Arguments

- Your program's main method is defined as:

```
public static void main(String [] args)
```

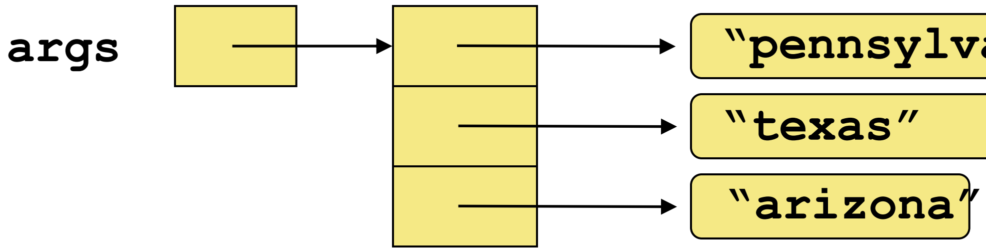
- The signature of the `main` method indicates that it takes an array of `String` objects as a parameter
- These values come from *command-line arguments* that are provided when the interpreter is invoked
- In Dr Java interactions pane, this invocation of the JVM passes three `String` objects (or tokens) as arguments to the `main` method of `StateEval`:

```
> java StateEval pennsylvania texas arizona
```

Command Line "Tokens"

Command Line Arguments

- These strings are stored at indexes `0-2` in the array `args` for the `main` method
- The array `args` will contain:



- Code in `main` can print the arguments:

```
for (String arg : args)
  System.out.println(arg);
```

The ArrayList Class

- The `ArrayList` class is in `java.util` package
- Instantiating an empty `ArrayList`

```
ArrayList<String> myList =
   new ArrayList<String>( );
```

- Like an array:
  - `ArrayList` can store a list of object references
  - You can access each one using a numeric index
- Unlike an array:
  - `ArrayList` object grows and shrinks as needed
  - You don’t use [ ] syntax with an `ArrayList` object
  - Cannot store primitive types (Use Wrapper classes)

The ArrayList Class

- The `ArrayList` class is available in the `java.util` package
- Instantiating an empty `ArrayList:`

```
ArrayList<String> myList =
   new ArrayList<String>( );
```

- An `ArrayList` stores references to the class inside the < > which allows it to store objects of that class only
- This is a part of Java’s generics capability which you will study further in CS210

The ArrayList Class

- Strings are inserted with a method invocation

```
boolean b = myList.add(string);   // to end
myList.add(index, string);  // at index
```

- When an element is inserted at a specific index, the other elements are "moved aside" to make room
- If `index` > `myList.size()`, the method throws an IndexOutOfBounds exception
- Elements are removed with a method invocation

```
String s = myList.remove(index);
```

- When an element is removed, the list "collapses" to close the gap and maintain contiguous indexes

ArrayList Efficiency

- The `ArrayList` class is implemented using an underlying array
- The array is manipulated so that indexes remain contiguous as elements are added or removed
- If elements are added to and removed from the end of the list, this processing is fairly efficient
- But as elements are inserted and removed from the front or middle of the list, the remaining elements are shifted

Javadoc

- Javadoc is a JDK tool that creates HTML user documentation for your classes and their methods
- In this case, user means a programmer who will be writing Java code using your classes
- You can access Javadoc via the JDK CLI:

```
> javadoc MyClass.java
```

- You can access Javadoc via Dr Java menu:

  Tools > Javadoc All Documents

  Tools > Preview Javadoc for Current Document

Javadoc

- The Javadoc tool scans your source file for specialized multi-line style comments:

```
/**
 * <p>HTML formatted text here</p>
 */
```

- Your Javadoc text is written in HTML so that it can appear within a standardized web page format

Block Tags for Classes

- At the class level, you must include these block tags *with data* (each on a separate line):

```
/**
  *  @author Your Name
  *  @version Version Number or Date
  */
```

- You should include HTML text describing the use of this class and perhaps give examples

Block Tags for Methods

- At the method level, you must include these block tags *with data* (each on a separate line):

```
/**
 *  @param HTML text for 1st parameter
 *  @param HTML text for 2nd parameter
 *  @return HTML text for return value
 */
```

- If there are no parameters or return type, you can omit these Javadoc block tags

In Line Tags

- At any point in your Javadoc HTML text, you may use In-Line Tags such as @link:

```
/**
 * <p>See website {@link name url}
 * for more details.</p>
 */
```

- In-Line tags are always included inside `{ }`
- These `{ }` are inside the `/**` and `*/` so the compiler does not see them

HTML Coding

- To the extent that time permits:
  - HTML Coding for text formatting
  - Questions on HTML and use in Javadoc