COMP 110/L Lecture 11

Kyle Dewey

Outline

- Command-line arguments and arrays
    - Array access
    - Array length
    - Array update
- `Integer.parseInt`

Command-Line Arguments

```
public class Foo {
  public static void
  main(String[] args) {
     ...
  }
}
```

–You’ve all seen code like this tons of times

```
public class Foo {
  public static void
  main(String[] args) {
    ...
  }
}
```

Command-line arguments

-The portion in red refers to the program's command-line arguments

```java
public class Foo {
  public static void
  main(String[] args) {
    ...
  }
}
```

Command-line arguments

```
javac Foo.java
java Foo one two
```

-The first line compiles your program (javac Foo.java)
-The second line runs your compiled program from the .class file generated (java Foo)

```
public class Foo {
  public static void
  main(String[] args) {
    ...
  }
}
```

Command-line arguments

```
javac Foo.java
java Foo one two
```

Command-line arguments

-The first line compiles your program (javac Foo.java)
-The second line runs your compiled program from the .class file generated (java Foo)
-The "one" and the "two" are command-line arguments
-In this case, there are two arguments: "one" and "two", respectively

Dissecting `String[] args`

- `String` refers to a single string
- `String[]` refers to an *array* of strings
  - Array: ordered, fixed-length list

Dissecting `String[] args`

- `String` refers to a single string
- `String[]` refers to an *array* of strings
  - Array: ordered, fixed-length list

```
javac Foo.java
java Foo one two
```

Dissecting `String[] args`

- `String` **refers to a single string**
- `String[]` **refers to an *array* of strings**
    - Array: ordered, fixed-length list

```
javac Foo.java
java Foo one two
```

`args`: **array of length 2**

First string: "`one`"

Second string: "`two`"

`java Foo one two`

`args`: array of length 2

First string: “`one`”

Second string: “`two`”

```
java Foo one two
```

`args`: array of length 2

First string: “`one`”

Second string: “`two`”

```
java Foo apple
java Foo one two
```

`args`: array of length 2

First string: "`one`"

Second string: "`two`"

```
java Foo apple
```

`args`: array of length 1

First string: "`apple`"

```
java Foo one two
```

`args`: array of length 2

First string: “`one`”

Second string: “`two`”

```
java Foo apple
```

`args`: array of length 1

First string: “`apple`”

```
java Foo foo bar baz
java Foo one two
```

`args`: array of length 2

First string: “`one`”

Second string: “`two`”

```
java Foo apple
```

`args`: array of length 1

First string: “`apple`”

```
java Foo foo bar baz
```

`args`: array of length 3

First string: “`foo`”

Second string: “`bar`”

Third string: “`baz`”

`java Foo foo bar baz`

`args`: array of length 3

First string: “`foo`”

Second string: “`bar`”

Third string: “`baz`”

`java Foo`

```
java Foo foo bar baz
```

`args`: array of length 3

First string: “`foo`”

Second string: “`bar`”

Third string: “`baz`”

```
java Foo
```

`args`: array of length 0

No contents.

Array Operations

Array Access

Can access array elements using square brackets ([ ]).

Need to access at a given *index*, starting from 0.

Array Access

Can access array elements using square brackets (`[]`).

Need to access at a given *index*, starting from 0.

```
args[0]
```

Array Access

Can access array elements using square brackets (`[]`).

Need to access at a given *index*, starting from 0.

```
args[0]
```

Accesses the element at index `0` (first element).

Array Access

Can access array elements using square brackets (`[]`).

Need to access at a given *index*, starting from 0.

```
args[0]
```

Accesses the element at index `0` (first element).

```
args[1]
```

Array Access

Can access array elements using square brackets (`[]`).

Need to access at a given *index*, starting from 0.

```
args[0]
```

Accesses the element at index `0` (first element).

```
args[1]
```

Accesses the element at index `1` (second element).

Array Access

Can access array elements using square brackets (`[]`).

Need to access at a given *index*, starting from 0.

```
args[0]
```

Accesses the element at index `0` (first element).

```
args[1]
```

Accesses the element at index `1` (second element).

```
args[x + 1]
```

Array Access

Can access array elements using square brackets (`[]`).

Need to access at a given *index*, starting from 0.

`args[0]`

Accesses the element at index `0` (first element).

`args[1]`

Accesses the element at index `1` (second element).

`args[x + 1]`

Accesses the element at whatever index `x + 1` evaluates to.

Example:
`PrintFirstThreeArgs.java`

Array Length

Can get the number of elements in the array as an `int` using `.length`

Array Length

Can get the number of elements in the array as an `int` using `.length`

```
java Foo one two
```

`args`: array of length 2

First string: “`one`”

Second string: “`two`”

Array Length

Can get the number of elements in the array as an `int` using `.length`

```
java Foo one two
```

`args`: array of length 2

First string: “`one`”

Second string: “`two`”

```
args.length // returns 2
```

Example: `ArgsLength.java`

Array Creation

Can create arrays of a given length using `new`

Array Creation

Can create arrays of a given length using `new`

```
int[] array = new int[2];
```

Array Creation

Can create arrays of a given length using `new`

```
int[] array = new int[2];
```

Creates an array of `int` holding two elements.

The two elements will both be `0`

Array Creation

Can create arrays of a given length using `new`

```
int[] array = new int[2];
```

Creates an array of `int` holding two elements.

The two elements will both be `0`

```
double[] array = new double[5];
```

Array Creation

Can create arrays of a given length using `new`

```
int[] array = new int[2];
```

Creates an array of `int` holding two elements.

The two elements will both be `0`

```
double[] array = new double[5];
```

Creates an array of `double` holding five elements.

The five elements will all be `0.0`

Array Creation

Can create arrays of a given length using `new`

```
int[] array = new int[2];
```

Creates an array of `int` holding two elements.
The two elements will both be `0`

```
double[] array = new double[5];
```

Creates an array of `double` holding five elements.
The five elements will all be `0.0`

```
long[] array = new long[0];
```

Array Creation

Can create arrays of a given length using `new`

```
int[] array = new int[2];
```

Creates an array of `int` holding two elements.

The two elements will both be `0`

```
double[] array = new double[5];
```

Creates an array of `double` holding five elements.

The five elements will all be `0.0`

```
long[] array = new long[0];
```

Creates an array of `long` holding zero elements.

AKA an empty array.

Array Update

Also use square brackets and indices to update an array.

Difference: array on the lefthand-side of the =

Array Update

Also use square brackets and indices to update an array.
Difference: array on the lefthand-side of the =

```
array[0] = 5;
```

Array Update

Also use square brackets and indices to update an array.

Difference: array on the lefthand-side of the =

```
array[0] = 5;
```

Sets value at index `0` of `array` to `5`

Array Update

Also use square brackets and indices to update an array.

Difference: array on the lefthand-side of the =

```
array[0] = 5;
```

Sets value at index `0` of `array` to `5`

```
array[20] = -7;
```

Array Update

Also use square brackets and indices to update an array.

Difference: array on the lefthand-side of the `=`

```
array[0] = 5;
```

Sets value at index `0` of `array` to `5`

```
array[20] = -7;
```

Sets value at index `20` of `array` to `-7`

Array Update

Also use square brackets and indices to update an array.
Difference: array on the lefthand-side of the =

```
array[0] = 5;
```

Sets value at index `0` of `array` to `5`

```
array[20] = -7;
```

Sets value at index `20` of `array` to `-7`

```
array[x + 1] = 8;
```

Array Update

Also use square brackets and indices to update an array.
Difference: array on the lefthand-side of the `=`

```
array[0] = 5;
```

Sets value at index `0` of `array` to `5`

```
array[20] = -7;
```

Sets value at index `20` of `array` to `-7`

```
array[x + 1] = 8;
```

Sets value at whatever index
`x + 1` evaluates to of `array` to `8`

Example:
`CreateArrayTwoElements1.java`

Another Way to Create Arrays

Can create an array and set initial values in a single expression via another form of `new`

Another Way to Create Arrays

Can create an array and set initial values in a single expression via another form of `new`

```
new int[]{42, 27}
```

Another Way to Create Arrays

Can create an array and set initial values in a single expression via another form of `new`

```
new int[]{42, 27}
```

Creates an array of length 2 with the contents `42`, `27`

Another Way to Create Arrays

Can create an array and set initial values in a single expression via another form of `new`

```
new int[]{42, 27}
```

Creates an array of length 2 with the contents `42,27`

```
new double[]{5.5}
```

Another Way to Create Arrays

Can create an array and set initial values in a single expression via another form of `new`

```
new int[]{42, 27}
```

Creates an array of length 2 with the contents `42,27`

```
new double[]{5.5}
```

Creates an array of length 1 with the contents `5.5`

Example:
`CreateArrayTwoElements2.java`

Arrays as Arguments

Arrays can be passed as method arguments just like any other type (the type is `int[]`, `double[]`, and so on).

Arrays as Arguments

Arrays can be passed as method arguments just like any other type (the type is `int[]`, `double[]`, and so on).

```
public static void method(int[] array) {
   ...
}
```

Arrays as Arguments

Arrays can be passed as method arguments just like any other type (the type is `int[]`, `double[]`, and so on).

```
public static void method(int[] array) {
  ...
}

public static void main(String[] args) {
  method(new int[]{1, 2});
}
```

Example:
`MethodPrintsFirstArrayElement.java`

`Integer.parseInt`

`Integer.parseInt`

- Allows for conversion from a `String` representing an integer to an `int`
- Useful for treating command-line arguments (which are always `String`) as `int`

Integer.parseInt

- Allows for conversion from a `String` representing an integer to an `int`
- Useful for treating command-line arguments (which are always `String`) as `int`

```
int x = Integer.parseInt(“42”);
// x now holds 42
```

`Integer.parseInt`

- Allows for conversion from a `String` representing an integer to an `int`
- Useful for treating command-line arguments (which are always `String`) as `int`

```
int x = Integer.parseInt("42");
// x now holds 42
```

```
int y = Integer.parseInt("128");
```

`Integer.parseInt`

- Allows for conversion from a `String` representing an integer to an `int`
- Useful for treating command-line arguments (which are always `String`) as `int`

```
int x = Integer.parseInt("42");
// x now holds 42
```

```
int y = Integer.parseInt("128");
// y now holds 128
```

Example:

`MultiplyFirstTwoArgs.java`