# COMP 110/L Lecture 17

Kyle Dewey

# Outline

- `String.length`
- `String.split`
- Multidimensional arrays

# `String.length`

Returns the number of `char`s in the given `String`

# `String.length`

**Returns the number of `char`s in the given `String`**

---

```
"abc".length()
```

# `String.length`

**Returns the number of `char`s in the given `String`**

```
"abc".length()
```

```
3
```

# `String.length`

**Returns the number of `char`s in the given `String`**

---

```
"abc".length()
```

```
3
```

---

```
"".length()
```

# `String.length`

**Returns the number of `char`s in the given `String`**

---

```
"abc".length()
```

```
3
```

---

```
"".length()
```

```
0
```

# Example:
`StringLength.java`

# String.split

# `String.split`

Allows for a `String` to be separated into different parts.

Returns an array of `String`s (`String[]`).

# `String.split`

Allows for a `String` to be separated into different parts.

Returns an array of `String`s (`String[]`).

---

```
"foo,bar".split(",")
```

# `String.split`

Allows for a `String` to be separated into different parts.

Returns an array of `String`s (`String[]`).

---

```
"foo,bar".split(",")
```

```
new String[]{"foo", "bar"}
```

# Example:
`SplitOnComma.java`

# What `split` Takes

`split` takes a *regular expression*.

Regular expressions describe different string patterns.

# What `split` Takes

`split` takes a *regular expression*.
Regular expressions describe different string patterns.

---

```
"foo,bar".split(",")
```

# What `split` Takes

`split` takes a *regular expression*.
Regular expressions describe different string patterns.

---

```
"foo,bar".split(",")
```

",": matches only one pattern: a comma

# What `split` Takes

`split` takes a *regular expression*.
Regular expressions describe different string patterns.

---

```
"foo,bar".split(",")
```

",": matches only one pattern: a comma

---

```
"foo.bar".split(".")
```

# What `split` Takes

`split` takes a *regular expression*.
Regular expressions describe different string patterns.

---

```
"foo,bar".split(",")
```

",": matches only one pattern: a comma

---

```
"foo.bar".split(".")
```

". ": matches **any** single character

# What `split` Takes

`split` takes a *regular expression*.
Regular expressions describe different string patterns.

---

```
"foo,bar".split(",")
```

",": matches only one pattern: a comma

---

```
"foo.bar".split(".")
```

". ": matches **any** single character

---

```
"foo.bar".split("\\.")
```

# What `split` Takes

`split` takes a *regular expression*.
Regular expressions describe different string patterns.

---

`"foo,bar".split(",")`

"," : matches only one pattern: a comma

---

`"foo.bar".split(".")`

" . ": matches **any** single character

---

`"foo.bar".split("\\.")`

"\\." : matches a period (backslash followed by a period)

# Example:

`SplitOnAnything.java`

# Regular Expressions

- Super popular for extracting values from `String` inputs
- Could easily spend a week on them
- Covered in later courses

# Multidimensional Arrays

# Recap - Arrays

Arrays are fixed-length sequences
of elements of the same type.

# Recap - Arrays

Arrays are fixed-length sequences
of elements of the same type.

```
new char[]{'a', 'b', 'c'}

new int[]{1, 2, 3}

new String[]{"foo", "bar"}

new double[]{1.2, 3.4}
```

# Multidimensional Arrays

Java also allows us to make arrays of *arrays*.
These are often called *multidimensional* arrays.

-"Multidimensional" because we need multiple dimensions to access any single element

# Multidimensional Arrays

Java also allows us to make arrays of *arrays*.
These are often called *multidimensional* arrays.

```
new int[][]{ new int[]{1, 2, 3},
             new int[]{4, 5},
             new int[]{6},
             new int[0],
             new int[]{7, 8, 9} }
```

-"Multidimensional" because we need multiple dimensions to access any single element
-This is specifically a two-dimensional array, since we need two dimensions to access a single int (specifically a row and a column)

# Multidimensional Arrays

Java also allows us to make arrays of *arrays*.
These are often called *multidimensional* arrays.

```
new int[][]{ new int[]{1, 2, 3},
             new int[]{4, 5},
             new int[]{6},
             new int[0],
             new int[]{7, 8, 9} }
```

Corresponding type: `int[][]`

-"Multidimensional" because we need multiple dimensions to access any single element
-This is specifically a two-dimensional array, since we need two dimensions to access a single int (specifically a row and a column)

# Multidimensional Array Utility

Commonly used for representing tables

# Multidimensional Array Utility

Commonly used for representing tables

| | | |
|---|---|---|
| 13 | 12 | 19 |
| 64 | 89 | 247 |
| 78 | 57 | 21 |

# Multidimensional Array Utility

Commonly used for representing tables

| | | |
|---|---|---|
| 13 | 12 | 19 |
| 64 | 89 | 247 |
| 78 | 57 | 21 |

```
new int[][]{ new int[]{13, 12, 19},
             new int[]{64, 89, 247},
             new int[]{78, 57, 21} }
```

# Accessing Rows

One row of  a two-dimensional array is an array...

# Accessing Rows

One row of  a two-dimensional array is an array...

```
int[][] array = ...;
int[] row = array[0];
```

# Accessing Rows

One row of  a two-dimensional array is an array...

```
int[][] array = ...;
int[] row = array[0];
```

# Accessing Columns

...and columns are individual elements of rows.

# Accessing Rows

One row of  a two-dimensional array is an array...

```
int[][] array = ...;
int[] row = array[0];
```

# Accessing Columns

...and columns are individual elements of rows.

```
int[][] array = ...;
int[] row = array[0];
int columnElement = row[5];
```

# Accessing Rows

One row of  a two-dimensional array is an array...

```
int[][] array = ...;
int[] row = array[0];
```

# Accessing Columns

...and columns are individual elements of rows.

```
int[][] array = ...;
int[] row = array[0];
int columnElement = row[5];
```

```
int[][] array = ...;
int columnElement = array[0][5];
```

-Last box is shorthand for the second box: we can access a row and a column element in a single expression

Example:

`AccessTwoDimensionalElement.java`