Strings

Christopher Simpkins
chris.simpkins@gatech.edu

`String` Values

Two ways to represent `String` values in a program:

- `String` literals

```
"foo"
```

- `String` variables

```
String foo = "foo";
```

`String` Concatenation

The `+` operator is overloaded to mean concatenation for `String` objects

- Strings can be concatenated

```
String bam = foo + bar + baz; // Now bam is "foobarbaz"
```

- Primitive types can also be concatenated with `Strings`. The primitive is converted to a String

```
String s = bam + 42; // s is "foobarbaz42"
String t = 42 + bam; // t is "42foobarbaz"
```

Note that `+` is only overloaded for `Strings`.

The `String` Class

`String` acts like primitive thanks to syntactic sugar provided by the Java compiler, but it is defined as a class in the Java standard library

- See `http://docs.oracle.com/javase/7/docs/api/java/lang/String.html` for details.
- Methods on objects are invoked on the object using the `.` operator

```
String empty = "";
int len = empty.length(); // len is 0
```

- Look up the methods `length`, `indexOf`, `substring`, and `compareTo`, and `trim`
- Because `String`s are objects, beware of null references:

```
int aPosInBoom = boom.indexOf("a"); // This won't compile
```

Putting it all together

Break out your laptops!