# Assess Yourself

Design a class, including attributes, method names, and constructors, for a *drinking glass*.

What things describe/define these objects?

What can an object of this class do? What can *other objects* do *with/to* this object?

# Assess Yourself

Firstly, we add the attributes:

```
public class Glass {
    // Easy stuff; primitive attributes
    public double height, radius;

    // More interesting; initialising attributes
    public double fillHeight = 0;

    public boolean isFull = false;

    // Even more interesting; using
    // other classes as attributes
    public Material material;

    public Shape shape;
}
```

# Assess Yourself

Next, we use the attributes to add a constructor:

```
public class Glass {
    // Note how we don't have to include the
    // variables we already initialised
    public Glass(double height, double radius,
                 Material material, Shape shape) {
        this.height = height;
        this.radius = radius;
        this.material = material;
        this.shape = shape;
    }
}
```

# Assess Yourself

We can also have multiple constructors:

```
public class Glass {
    public Glass(double height, double radius,
            Material material, Shape shape, double fillHeight) {
        this.height = height;
        this.radius = radius;
        this.material = material;
        this.shape = shape;
        this.fillHeight = fillHeight;

        // Note: approxEqual needs to be written by us
        if (approxEqual(this.height, this.fillHeight)) {
            this.isFull = true;
        }
    }
}
```

# Assess Yourself

Finally, we add methods so our objects can do things:

```
public class Glass {
    public double calculateVolume() {
        return Math.PI * radius * radius * height;
    }

    public void fillGlass() {
        fillHeight = height;
        isFull = true;
    }

    public void emptyGlass() {
        fillHeight = 0;
        isFull = false;
    }
}
```

# Assess Yourself

Finally, we add methods so our objects can do things:

```
public class Glass {
    // Note: it would be more appropriate to define
    // this in a "utility" class
    public static boolean approxEqual(double var1,
                                      double var2) {
        return Math.abs(var1 - var2) < 0.00001;
    }

    public static String message() {
        return "I am a Glass";
    }
}
```

SWEN20003
Object Oriented Software Development

# Classes II

Semester 1, 2019

# The Road So Far

**Lectures**

- Welcome to SWEN20003
- Classes and Objects

**Tutorials**

- Introduction/meet and greet
- Programming revision
- Classes and Objects

# Lecture Objectives

After this lecture you will be able to:

- Define more complex classes, that use other classes as attributes
- Use Strings to store text
- Use the primitive "wrapper" classes
- Use *methods* to manipulate Strings
- Create well-formatted Strings

# Motivating Example

You are writing a simple game with the Slick 2D game library, where you control a plane and fly it around the screen.

Implement a class for the plane: what attributes and methods might it have?

**More details:**
The plane is represented by a 2D bitmap image. It flies around the screen at 0.04 pixels/second, and rotates at 10 degrees/second.

# Assess Yourself

```
public class Plane {
    // Constants
    public static final double VELOCITY = 0.04;
    public static final double ROTATION_SPEED = 10.0;

    // Instance variables
    public double x, y;
    public boolean isMoving;
    public Image image;

    // One potential design
    public static final String IMAGE_LOCATION = "plane.jpg";
}
```

But what is a `String`?

# Assess Yourself

A "String" is a(n) what?

1. Object (not technically correct)
2. Class
3. Variable
4. Data Type
5. Method
6. Privacy Modifier
7. I have literally no clue

# Strings

- Strings store sequences of characters
- Are a Java class
- Used to represent messages, errors, and "character" related attributes like Name
- Incredibly powerful for input and output

**Keyword**

*String:* A Java class made up of a sequence of characters.

# Strings

Some examples of String variables

```
String s1 = "This is a String";
String s2 = "This is " + "also a String";
String s3 = "10";
String s4 = "s3 is still a string, even though it's a number";
```

Java Strings are almost identical to Python, except you **can't use single quotes**.

# Assess Yourself

What does this code output?

```
System.out.println(""Low-key" doesn't mean what you think it does.");
```

1. ""Low-key" doesn't mean what you think it does."
2. "Low-key" doesn't mean what you think it does.
3. Low-key doesn't mean what you think it does.
4. Error

# Special Characters

- Some characters (like ") are "reserved"
- Mean something special to Java
- Need to "escape" them with "\" to use alternate meaning
- Examples "\n" (newline), "\t" (tab) "\"" (quotation)

```
System.out.println("\"Lowkey\" doesn't mean what you think it does.");
```

## Keyword

*Escaping:* To include "special" characters in a string, use "\" to *escape* from that character's normal meaning.

# String Operations

- You can use + (and +=) to append/concatenate two strings
  - `System.out.println("Hello " + "World");`
  - Prints `"Hello World"`
- + is clever: if either operand is a string, it will turn the other into a string
  - `System.out.println("a = " + a + ", b = " + b);`
  - If a = 1 and b = 2, this prints: `"a = 1, b = 2"`
- Why is this useful?

# Assess Yourself

- `System.out.println("1 + 1 = " + 1 + 1);`
- Actually prints `"1 + 1 = 11"`
- `System.out.println("1 + 1 = " + (1 + 1));`
- Prints `"1 + 1 = 2"`

# Assess Yourself

- Name some "logical" things you might do with a String
- Think about how you would do them in C and Python

# String Methods

- Length
  - C: Need a helper/buddy variable
  - Python: `len("Hello")`
  - Java: `"Hello".length()`
- Upper/Lower case
  - C: `toupper(*s)`
  - Python: `s.upper()`
  - Java: `s.toUpperCase()`
- Split
  - C: Stop
  - Python: `s.split()`
  - Java: `s.split(" ")`

# String Methods

- Check substring presence
  - C: Why
  - Python: `"Hell" in s`
  - Java: `s.contains("Hell")`
- Find substring location
  - C: Never mind
  - Python: `s.find("Hell")`
  - Java: `s.indexOf("Hell")`
- Substring
  - C: I'm out
  - Python: `s[2:7]`
  - Java: `s.substring(2, 7)`

# String Methods

- The full String class documentation can be found here.

# Assess Yourself

What does this output?

```
String s = "Hello World";
s.toUpperCase();
s.replace("e", "i");
s.substring(0, 2);
s += " FIVE";
System.out.println(s);
```

"Hello World FIVE"

# Immutability

Our first "object oriented" concept:

- `Strings` are immutable; once created, they can't be modified, only replaced
- This means that every `String` operation **returns** a new `String`
- We'll look at immutability in more detail soon...
- Let's fix up that code

# Assess Yourself

What does this output?

```
String s = "Hello World";
s = s.toUpperCase();
s = s.replace("e", "i");
s = s.substring(0, 2);
s += " FIVE";
System.out.println(s);
```

"HE FIVE"

# Assess Yourself

What does this output?

```
System.out.println("Hello" == "Hello");
```

`true`

# Assess Yourself

What does this output?

```
String s = "Hello";
System.out.println(s == "Hello");
```

true

# Assess Yourself

What does this output?

```
String s = "Hello";
String s2 = "Hello";
System.out.println(s == s2);
```

`true`

# Assess Yourself

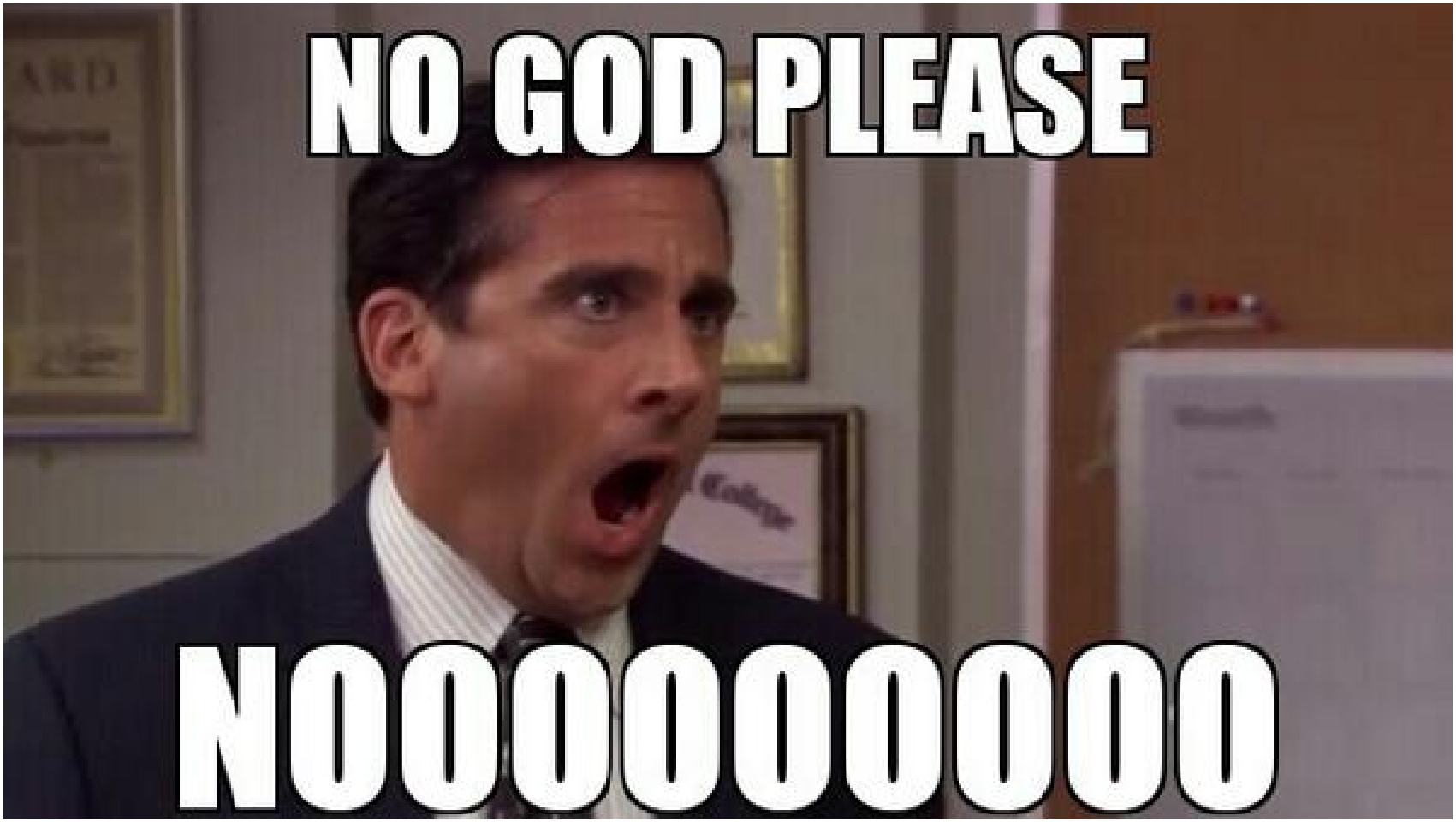
What does this output?

```
String s = "Hello";
String s2 = new String("Hello");
System.out.println(s == s2);
```

`false`

Guess what... Pointers are back!

# Equality



# Equality

- All classes in Java are actually pointers, or references
- An important point when we discuss privacy and mutability
- To check equality between two *objects*

```
String s = "Hello";
String s2 = new String("Hello");
System.out.println(s.equals(s2));
```

```
true
```

## Keyword

*.equals:* A method used to check two *objects* for equality

# “Primitive Classes”

- Java has a number of “pre-packaged” classes
- `String` is the most important for us right now
- But what else is there?

# Motivating Example

- Numbers (etc.) are automatically converted to Strings when needed
- What about the other way around?
- ~~`int x = "10"`~~
- Need to find another way

# Wrapper Classes

- Java provides "wrapper" classes for primitives
- Allows primitives like `int` and `double` to be "packaged" or "boxed" into objects
- Allows primitives to "pretend" that they are classes (this is important later)
- **Provides extra functionality for primitives**

## Keyword

*Wrapper:* A class that gives extra functionality to primitives like `int`, and lets them behave like objects

# Wrapper Classes

| Primitive | Wrapper Class |
|---|---|
| `boolean` | Boolean |
| `byte` | Byte |
| `char` | Character |
| `int` | Integer |
| `float` | Float |
| `double` | Double |
| `long` | Long |
| `short` | Short |

# Integer Class

Provides a number of methods such as:

- Reverse: `Integer.reverse(10)`
- Rotate Left: `Integer.rotateLeft(10, 2)`
- Signum: `Integer.signum(-10)`
- **Parsing**: `Integer.parseInt("10")`

```
Integer x = Integer.parseInt("20");
int y = x;
Integer z = 2*x;
```

# Parsing

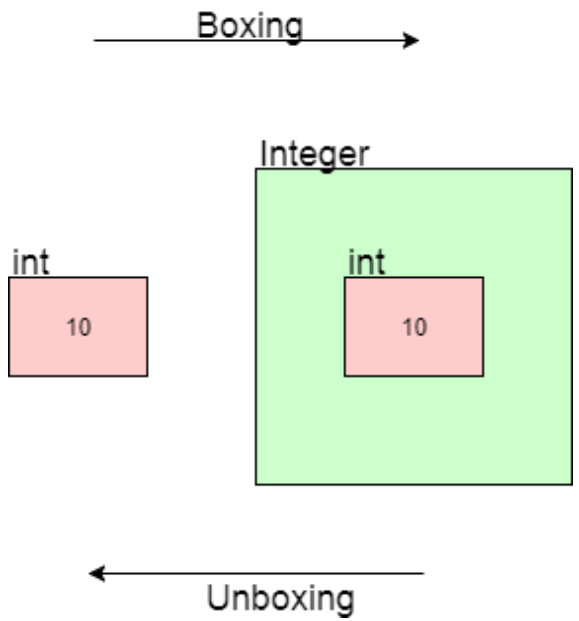
Every wrapper class has a parse function:

- xxx var = XXX.parseXXX(<string>);
- `int i = Integer.parseInt("1");`
- `double d = Double.parseDouble("1");`
- `boolean b = Boolean.parseBoolean("TruE");`

**Keyword**

*Parsing:* Processing one data type into another

# Automatic Boxing/Unboxing



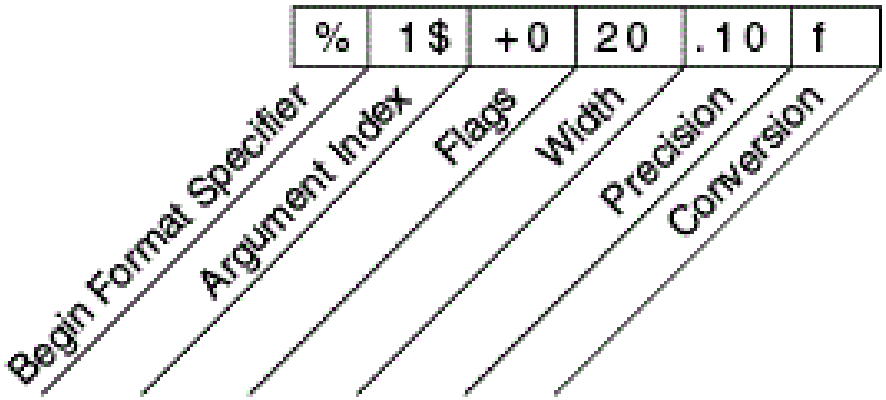

## Keyword

*(Un)Boxing:* The process of converting a primitive to/from its equivalent wrapper class

# Formatting

- Java allows you to format Strings much like C and Python
- Use `String.format(...)` to create a String to store in a variable
- Use `System.out.format(...)` to write formatted output
- The full documentation for formatting can be found here

# Formatting

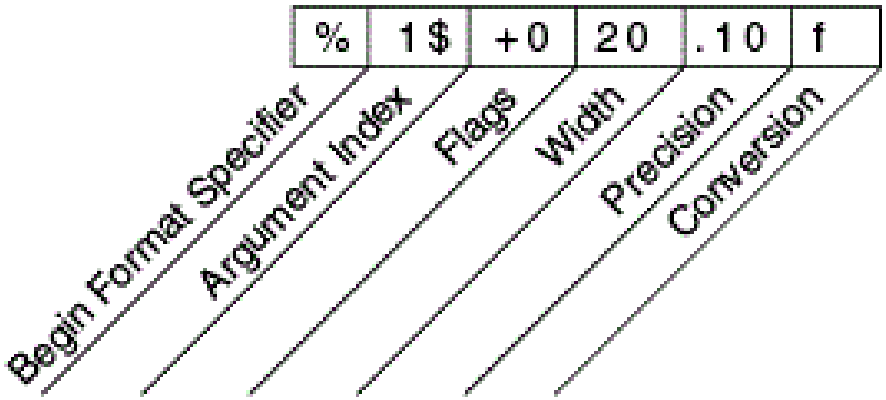



Examples:

```
String s = String.format("%3.2f", 4.56789);
```

```
"4.57"
```

# Formatting

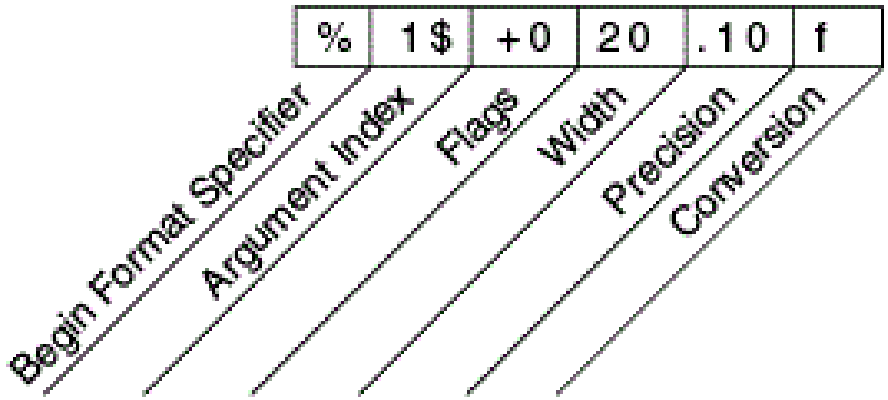



Examples:

```
String s = String.format("%+05d", 10);
```

"+0010"

# Formatting




Examples:

```
System.out.format("%2$d %<05d %1$d %3$10s", 10, 22, "Hello");
```

```
"22 00022 10       Hello"
```

# Metrics

Write a program that can take the following number, and output it in the form below.

```
3.14159265
```

```
"The value of 'pi' to four places is 0003.1416"
```

# Metrics

Write a class to represent a book. It can have whatever instance variables you see fit (title, author, etc.) but it must have a variable called `contents`.

Add at least one constructor, and separate methods to extract the first, last, and middle words of the book's text, in upper case.

Use any approach you like to find the middle word, or better yet, try multiple approaches.

# Metrics

An example of how we might use this class:

```
Book book = new Book("Game of Thrones", "Winter is coming!");
System.out.println(book.firstWord());
System.out.println(book.middleWord());
System.out.println(book.lastWord());
```

```
"WINTER"
"IS"
"COMING!"
```