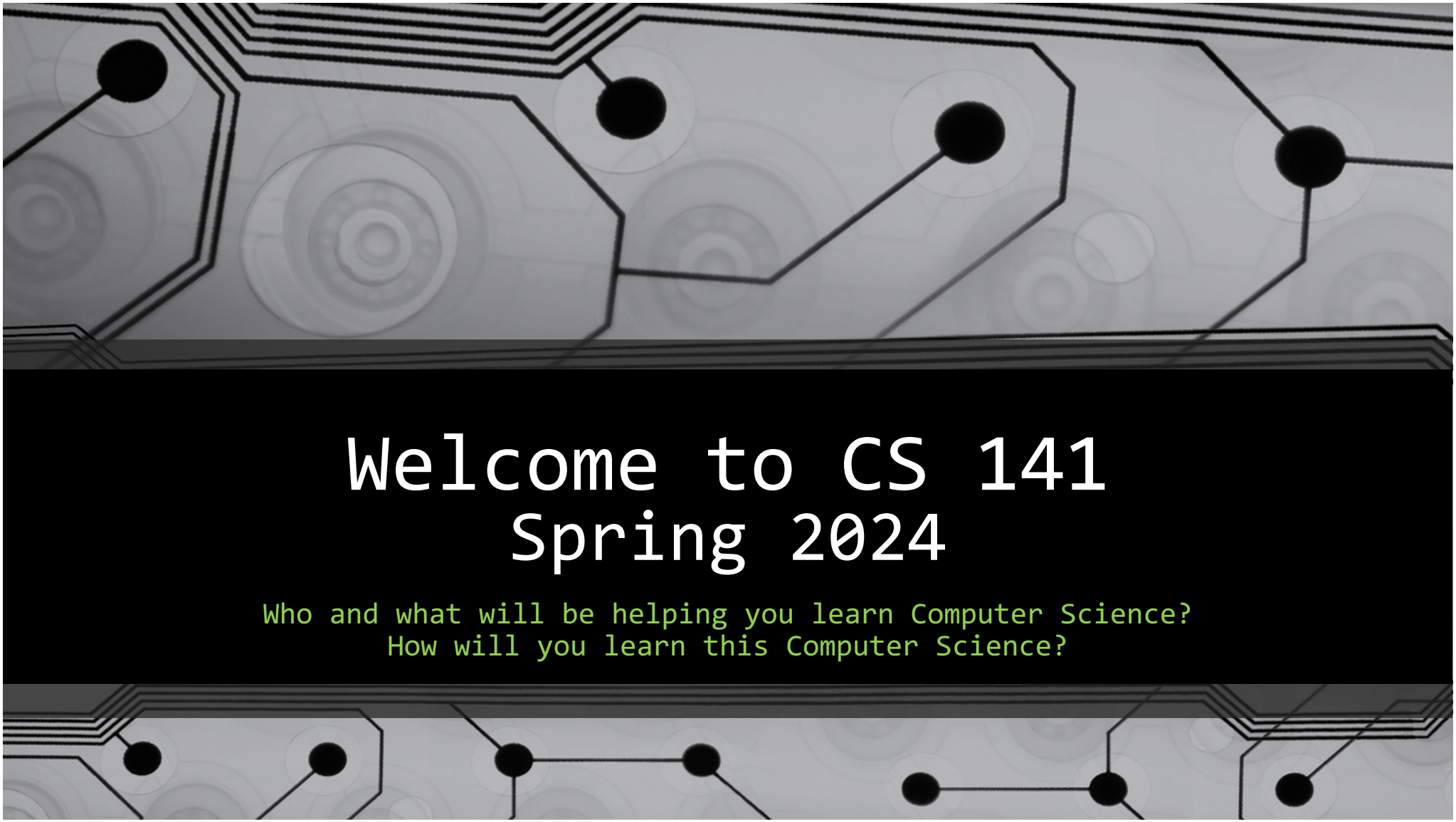


Welcome to CS 141
Spring 2024

Who and what will be helping you learn Computer Science?
How will you learn this Computer Science?

What we're doing

This course teaches problem solving on a computer. We will be using the Java programming language, but the point of the class is **not** just to learn Java. Rather, you are learning how to express what you want the computer to do, with Java being the specific language you're using. My goal is to teach programming as a tool for use in Computer Science or your own field.

People

Professors:
David Bunde (dbunde@knox.edu, SMC E203)
Michael Gerten (mgerten@knox.edu, SMC E206)

TAs: Cole Andrews, Fazil Kagdi, Tim Lorenz, Eita Ueda

Can also get questions answered by Red Room tutors, others who have taken the class…

Grading and stuff

45% for exams (in class, on paper)
- 2 during the term and a final, 15% each
- all are open notes (no computer use allowed)

10% for participation in “iClicker questions”

20% for questions in interactive textbook
- 10% for participation activities
- 10% for challenge activities

25% homework:
- mostly week-long programming projects, started during the weekly lab

Labs on Tuesday

- Lab period on Tuesday is for getting started with new material when help is easily available
  - You'll be starting your HW assignment for the week
- It will generally and most likely take you a lot more than 1 hour to complete
  - Whatever you don't finish becomes your homework for the week
  - Finish each lab by the next Tuesday
- We will check in with each student during the lab
  - To make sure you can get started, that you have things installed, that you make some progress

From Paulo Freire's *Pedagogy of the Oppressed*

> *Education thus becomes an act of depositing, in which the students are the depositories and the teacher is the depositor. Instead of communicating, the teacher issues communiques and makes deposits which the students patiently receive, memorize, and repeat. This is the "banking' concept of education, in which the scope of action allowed to the students extends only as far as receiving, filing, and storing the deposits.*

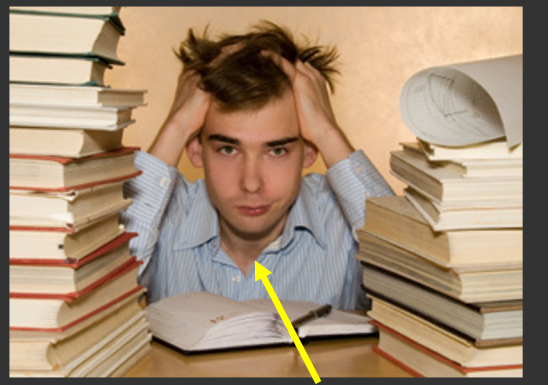
There is lots of evidence that the "banking deposit model" of education works very poorly!

This class is (mostly) a “flipped class”

We are moving what usually happens inside class (lecture on the basic outline of things) to before class,

and the stuff that usually happens outside class (homework covering harder stuff) into class.

Why are we doing this?



Homework looks like this!
I’m alone and this work is tough!

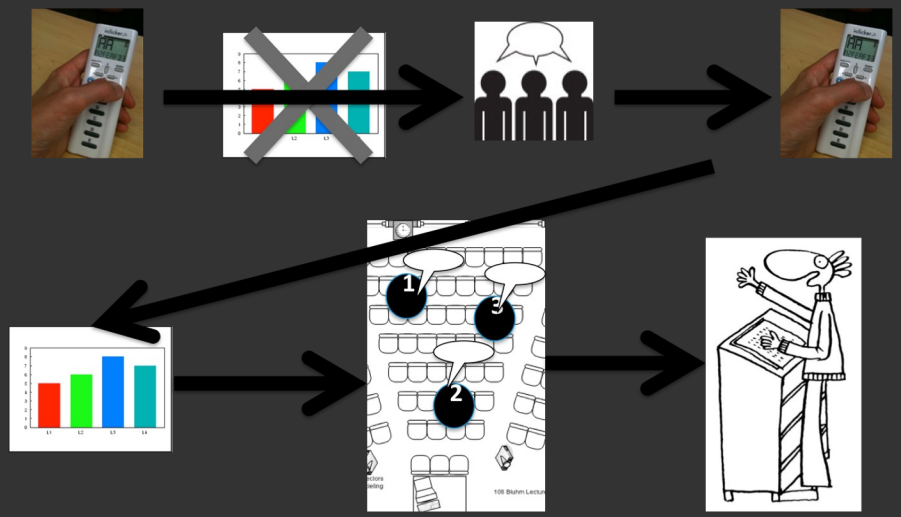
Class looks like this!
Everyone is together but we’re learning the easier basic stuff!

What does learning look like?

- Learning takes effort, practice, discussion, reflection
- Learning is *active*, not *passive*
- Learning is sometimes solitary, but is often social
- Education is a *process*, not a *result*
- Knowledge is not knowing the right answer,
  but rather knowing how to construct new knowledge

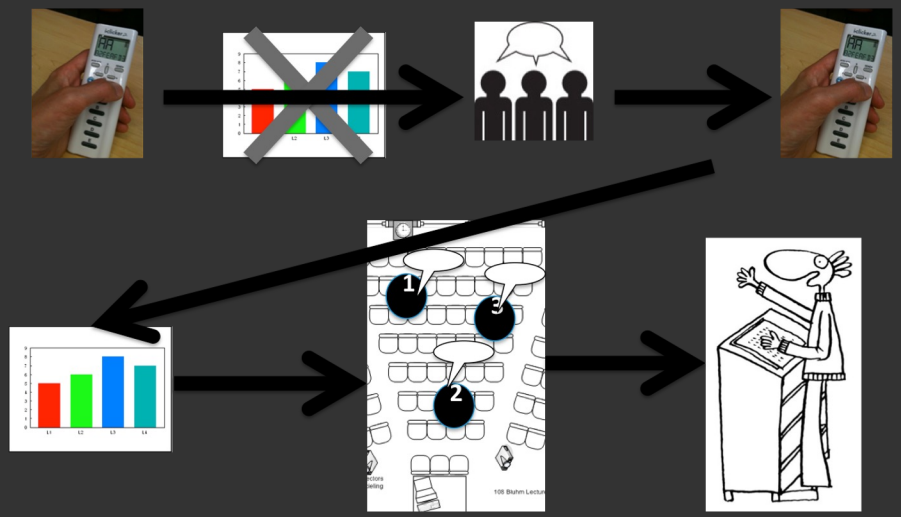
iClicker question workflow

- Answer for yourself (1-2 mins)
- Discuss with small group (2-3 mins)
- Answer again based on consensus of group
- See how everyone else voted
- Go over each answer
- Answer questions
- Mini-lecture



iClicker question workflow

- Answer for yourself (1-2 mins)
- Discuss with small group (2-3 mins)
- Answer again based on consensus of group
- See how everyone else voted
- Go over each answer
- Answer questions
- Mini-lecture



"iClicker" questions are graded on **participation ONLY**!

You can miss **3 classes** without losing any points on your clicker questions.

Now let’s practice
“iClicker” questions!

Which of the following will most contribute to your learning in this class?

A. Reading interactive textbook

B. Attending and participating in class

C. Completing lab assignments

D. Studying for and taking tests

E. Asking questions in office hours and Red Room

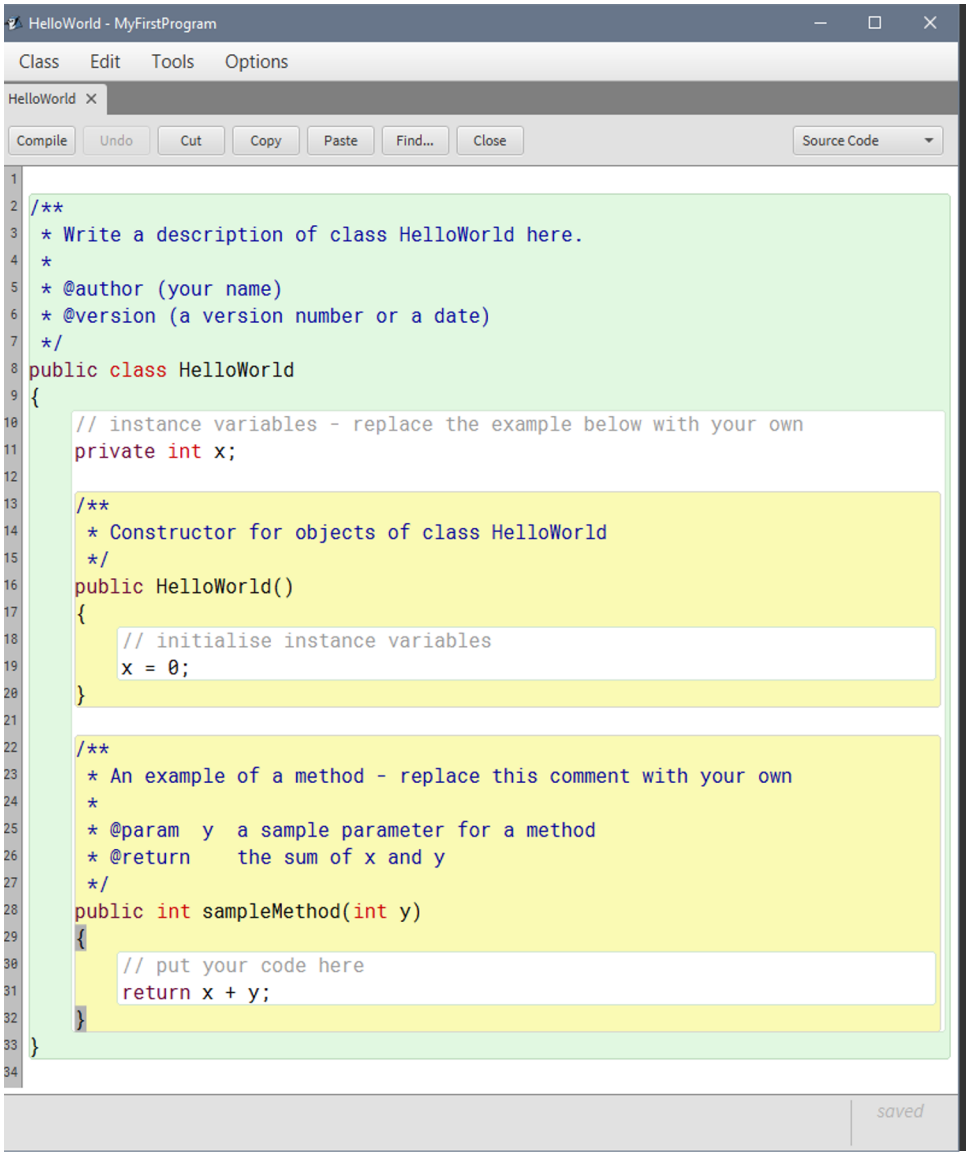
We are going to program in Java

Programming: writing out sequences of commands for the computer to carry out

Java is a high-level programming language

believe it or not, there are lower-level languages that are much harder for humans to read

DEMO of first program in BlueJ IDE



Big Oof…

Okay… BlueJ is trying to be helpful but that's a little too much too fast.

HelloWorld - MyFirstProgram

Class Edit Tools Options

HelloWorld

Compile Undo Cut Copy Paste Find... Close Source Code

```
/**
 * Write a description of class HelloWorld here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class HelloWorld
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class HelloWorld
     */
    public HelloWorld()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     *
     * @param  y  a sample parameter for a method
     * @return    the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

saved

HelloWorld - MyFirstProgram

Class Edit Tools Options

HelloWorld

Compile Undo Cut Copy Paste Find... Close Source Code

```
public class HelloWorld {

}
```

1

HelloWorld - MyFirstProgram

Class Edit Tools Options

HelloWorld

Compile Undo Cut Copy Paste Find... Close Source Code

```
/* Author:   Vera Kazakova
   date:     September, 2021
   program:  my very first program that prints "Hello, World!")
*/
public class HelloWorld {
    public static void main(String[] args) {

    }
}
```

2

HelloWorld - MyFirstProgram

Class Edit Tools Options

HelloWorld

Compile Undo Cut Copy Paste Find... Close Source Code

```
/* Author:   Vera Kazakova
   date:     September, 2021
   program:  my very first program that prints "Hello, World!")
*/
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");//print greeting
    }
}
```

3

BlueJ: Terminal Window - MyFirstProgram

Options

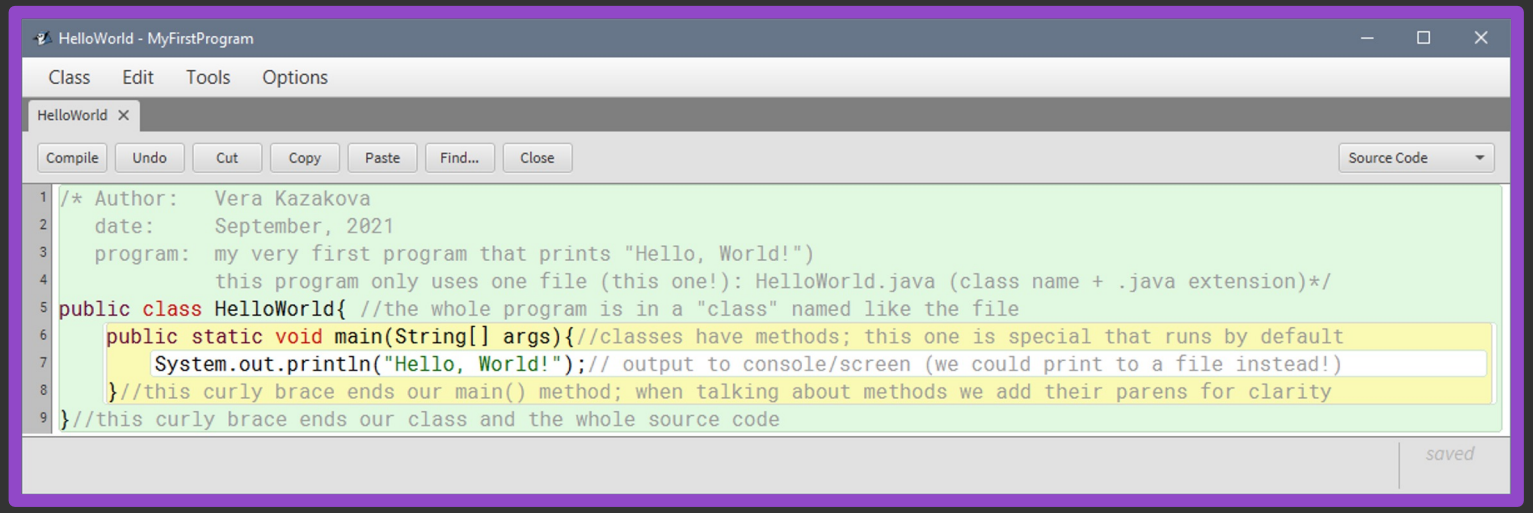
```
Hello, World!
```

Can only enter input while your programming

4

The actual code is just one line:

```
System.out.println("Hello, World!");
```



Function with a name and an input like f(x)

- f is System.out.println
- x is “Hello, World!”
- We are passing x as input to f(), to get f(x)
- Similarly, we are passing “Hello, World!” as input to System.out.println( )

You must unlearn what you have learned about grammar

Computer programming languages use grammatical symbols as pure *symbols* that mean different things in a programming context than they would in plain English.

This may contradict everything you've learned about English grammar your whole life.

Computer Science is not commonly taught in high school

Most students have a general idea of what math is,
what chemistry is,
what history is,
…

Computer Science? Not so much.

Some conventions

- **we count from zero:**

  first element is 0^{th} element, it is at position zero (this will matter later)

- **we follow language-specific syntax**
  - Java statements end with a semicolon
  - classes, methods, if statements, and loops are surrounded by curly braces { } ← we'll discuss this more later

- **how code looks matters**
  - bad looking code may break your program (python)
  - if it doesn't break your program, it will still break your programming/debugging
  - so what's some good looking code?
    - meaningful variable names
    - clear helpful comments
    - consistent and correct indentation
    - consistent spacing meaningfully grouping chunks of code

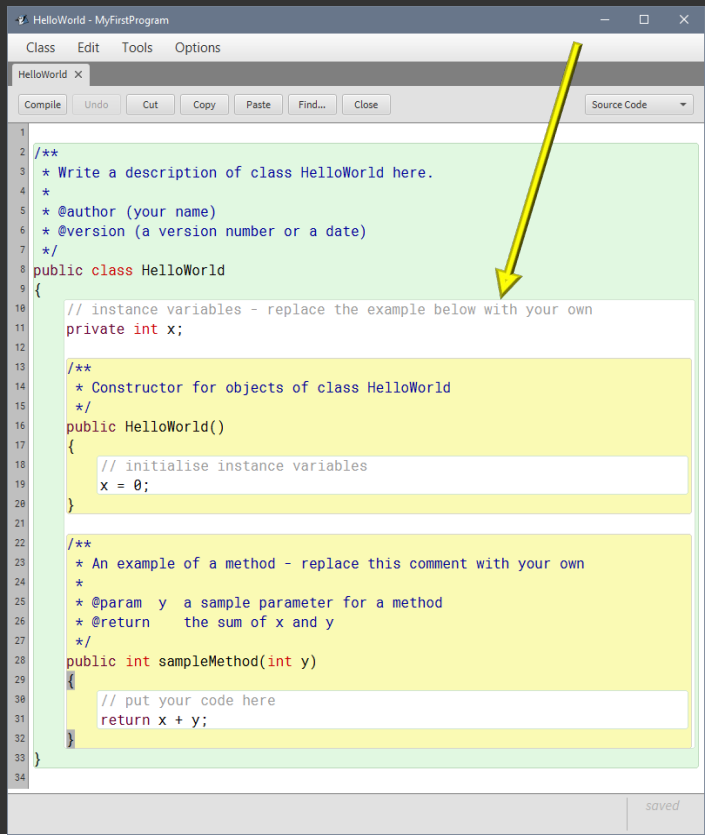
Some conventions

- **we count from zero:**

  first element is 0th element, it is at position zero (this will matter later)

- **we follow language-specific syntax**
  - Java statements end with a semicolon
  - classes, methods, if statements, and loops are surrounded by curly braces { }
- **how code looks matters**
  - bad looking code may break your program (python)
  - if it doesn't break your program, it will still break your programming/debugging
  - so what's some good looking code?
    - meaningful variable names
    - clear helpful comments
    - consistent and correct indentation
    - consistent spacing meaningfully grouping chunks of code

Turns out that was actually some great looking code!



Variables: containers for our data

If we couldn’t store values with a programming language,
it would be difficult to do any computation:

- calculate things
- recall stored data
- change data
- etc.

Java is strongly-typed

- strongly-typed :
  you need to declare the datatype of each variable when you create it

- We start with 3 datatypes:
  - String (capitalization matters in Java)
  - int
  - double

datatypes

- **String** can store strings of characters
  - can be as long as the computer has memory to hold!
  - MUST start with a capital; it's special → a class; we'll discuss this more later

- **int** can store integers from -2 billion to 2 billion, approximately
  - larger numbers are stored in a separate type called "long"
  - we won't use anything larger than an int in CS-141

- **double** can store decimal numbers
  - double is short for "double-precision floating point"
  - floating-point basically means "scientific notation" where we "float" the decimal point around by changing exponents (you don't need to know that, it's just trivia)

datatypes

What's up with that capitalization?
primitive types → lowercase
reference types → uppercase

- **String** can store strings of characters
  - can be as long as the computer has memory to hold!
  - MUST start with a capital; it's special → a class; we'll discuss this more later

- **int** can store integers from -2 billion to 2 billion, approximately
  - larger numbers are stored in a separate type called "long"
  - we won't use anything larger than an int in CS-141

- **double** can store decimal numbers
  - double is short for "double-precision floating point"
  - floating-point basically means "scientific notation" where we "float" the decimal point around by changing exponents (you don't need to know that, it's just trivia)

variables of type **String**

String holds characters, such as “It’s over 9000!!!”

```
String myString = “Hello, World!”;//make a new String with a value
myString = myString + “!!”; //we change that value

//we can print literals and variables to screen
System.out.print(“My string was: ” + myString + “ Over and out.”);
```

Output: My string was: Hello, World!!! Over and out.

note on syntax:
String variables (mystring) and String literals (“Hello, World!”) are both of type String
String types can be concatenated with +

variables of type int

`int` holds integer values (… -2, -1, 0, 1, 2, …)

```
public static void main(String[] args) {
    int x = 0;   //initialized on declaration with a literal value
    int y;       //declared but not initialized
    y = 0;       //now 'y' holds a value
}
```

note on syntax:
we can declare and initialize
or
declare only and initialize later

variables of type **double**

```
double d1 = 6.725; // double can store decimal numbers

double d2=6; // ints can be “promoted” to double
```

note on syntax:
space after type is mandatory, while space between (variable or literal) and (operand or semicolon) is optional
(just try to be consistent, for aesthetic reasons)

Our programs and variables need names, **identifiers**, so we can refer to them as needed

- must start with a letter (upper or lowercase) or _ (underscore)
- followed by zero or more letters, digits, _
- variable names start with lower case
- uppercase is reserved for classes (such as `class MyProgram` and `String`); we'll talk more about classes a bit later in the course)

```
1  class MyProgram {
2    public static void main(String[] args) {
3      int myInt = 5;
4      String greeting = "Hello, World!";
5      System.out.println(greeting);
6    }
7  }
```

Which of these are not legal identifiers?

A: FiveGuysBurgerCalorieCalculator
B: 5GuysBurgerCalorieCalculator
C: Five_guys_burger_calorie_calculator
D: all of the above
E: none of the above

Which of these are not legal identifiers?

A: FiveGuysBurgerCalorieCalculator
B: 5GuysBurgerCalorieCalculator Identifiers cannot start with a digit!
C: Five_guys_burger_calorie_calculator
D: all of the above
E: none of the above

Which of these are not legal identifiers?

A: DR_WHO
B: _DR_WHO_
C: DR WHO
D: all of the above
E: none of the above

Which of these are not legal identifiers?

A: DR_WHO
B: _DR_WHO_
C: DR WHO Identifiers cannot have spaces!
D: all of the above
E: none of the above

Which of these are not legal identifiers?

A: smileyFace:)
B: smiley:)Face
C: :)smileyFace
D: all of the above
E: none of the above

Which of these are not legal identifiers?

A: smileyFace:)
B: smiley:)Face
C: :)smileyFace
D: all of the above
E: none of the above

Identifiers can only contain letters, digits, and underscores.

mixing doubles and ints

```
// will this work?
int x = 6.7;
```

A: yes
B: no
C/D/E: I have no idea

mixing doubles and ints

```
// will this work?
int x = 6.7;
```

A: yes
B: no This will not work because 6.7 is not an int.
C/D/E: I have no idea

mixing doubles and ints

```
// will this work?
int x = 6.0;
```

A: yes
B: no
C/D/E: no idea

mixing doubles and ints

```
// will this work?
int x = 6.0;
```

A: yes
B: no
C/D/E: no idea

It doesn't matter that 6.0 is technically 6; for safety reasons, Java will never convert a double to an int unless you explicitly say it's OK with a (*cast)*

Let's see how next!

Casting: turning one type into another

Widening Casting or Type Promotion (automatic): converting a larger type; no loss of precision

byte → short → char → int → long → float → double

```
double myDouble = 5; //auto promoted to 5.0
```

Narrowing Casting or Downcasting (manual via parens): converting to a smaller type; loss of precision)

double → float → long → int → char → short → byte

```
double myDouble = 6.7;
int myInt = (int)myDouble; // manually downcast to 6
```

mixing doubles and ints

```
// will this work?
int x = (int)6.0;
```

A: yes
B: no
C/D/E: no idea

mixing doubles and ints

```
// will this work?
int x = (int)6.0;
```

A: yes We are forcing a manual downgrade of double to int.
B: no
C/D/E: no idea

What will this print?

```
int x = (int)6.7;
System.out.println(x);
```

A: 6
B: 7
C: 6.0
D/E: no idea

What will this print?

```
int x = (int)6.7;
System.out.println(x);
```

A: 6
B: 7
C: 6.0
D/E: no idea

The (int) part is a *cast* that forces a conversion.
Casts *truncate* (i.e. round down).

Operators let us operate on variables

so far we've seen the Assignment operator **=**

```
int x = 5;
```

and the typecasting operator **(datatype)**

```
int x = (int)5.0;
```

Arithmetic operations

All of the arithmetic operations you learned in math also work in Java

```
+ - * /
```

What does this print?

```
int x = 4 + 3 * 6;
System.out.println(x);
```

A: 42
B: 22
C: 0
D: x
E: none of the above

What does this print?

```
int x = 4 + 3 * 6;
System.out.println(x);
```

A: 42
B: 22
C: 0
D: x
E: none of the above

Order of operations works the same way as in math!

* and / before + and -

PEMDAS:
parentheses,
exponents,
multiplication/division,
addition/subtraction

What does this print?

```
int num = 4 / 5 * 100 + 2;
System.out.println(num);
```

A: 2
B: 22
C: 42
D: 82
E: none of the above

What does this print?

```
int num = 4 / 5 * 100 + 2;
System.out.println(num);
```

A: 2 **4 / 5 = 0 (integer division!)**
B: 22
C: 42
D: 82
E: none of the above

What does this print?

```
double a = 5;
double b = a / 10 * 20;
System.out.println(b);
```

A: 0.0
B: 5.0
C: 10.0
D: 0.05
E: none of the above

What does this print?

```
double a = 5;
double b = a / 10 * 20;
System.out.println(b);
```

A: 0.0
B: 5.0
C: 10.0
D: 0.05
E: none of the above

What does this print?

```
double d = 8 / 10;
System.out.println(d);
```

A: d
B: 0.8
C: 0
D: 2
E: it will crash and throw an exception

What does this print?

```
double d = 8 / 10;
System.out.println(d);
```

A: d
B: 0.8
C: 0
D: 2
E: it will crash and throw an exception

8 and 10 are both ints, so clearly you must have wanted Integer division!
10 divides into 8 zero times!

LHS assignment portion is independent of the RHS calculation portion.

How can we get 0.8?

A: double d = 8.0 / 10.0;
B: double d = 8.0 / 10;
C: double d = 8 / 10.0;
D: all of the above
E: none of the above

How can we get 0.8?

A: double d = 8.0 / 10.0;
B: double d = 8.0 / 10;
C: double d = 8 / 10.0;
D: all of the above
E: none of the above

Increment operators: ++, +=

Java has a lot of shorthand operations.

If x is an int:

```
x++;
x+=1;
x=x+1;
```

these all do the same thing:
increment x by 1

Increment operators: --, -=

Java has a lot of shorthand operations.

If x is an int:

```
x--;
x-=1;
x=x-1;
```

these all do the same thing:
decrement x by 1

Have you heard of the programming language C++?

This is a pun of sorts, because C++ is an improvement to C.

So they did a ++ of C and got C++.

This is what passes for humor around here!

mod or modulo operation

We saw that using / with integers performs *integer division*, which only keeps the quotient and ignores the remainder.

What if we want to keep the remainder?

We can use the % sign to perform the *mod* operation, which returns only the remainder.

7.0 / 4 is 1.75 (because 4 is auto-promoted to double)
7 / 4 is 1 (because of integer division)
7 % 4 is 3 (because 4 goes into 7 one time, with a remainder of 3)

mod

What is 6 % 8?

A: 0
B: 1
C: 2
D: 6
E: 8

mod

What is 6 % 8?

A: 0
B: 1
C: 2
D: 6 8 goes into 6 zero times, with a remainder of 6
E: 8

X % 7

We don't know what X is, but we know it’s positive.
What is the maximum possible result?

A: 0
B: 1
C: 6
D: 7
E: 8

X % 7

We don't know what X is, but we know it's positive.
What is the maximum result?

A: 0
B: 1
C: 6
D: 7
E: 8

If we mod by 7, we get values from 0 to 6.
We could never get a result of 7
(having a remainder of 7, we could have just added 1 to the quotient, going back to a remainder of 0)

DONE !