

# Computer Programming in Java

## Bush School *CPJava Fall 2022*

### *Block D*



Welcome to CPJava!

Dru & not Gru!

*Chandru Narayan*  
*Bicyclist Astronomer Engineer*  
*Teaching Astronomy and CS*



# Introductions!

1. State your name - how would you like to be addressed?
2. Your personal pronoun?
3. What are your hobbies?
4. Say something interesting or peculiar about yourself
5. Do you have any exposure to programming?
6. What made you choose this course?
7. What are your expectations from this course?
- 8.

# A unique first course in programming with some advanced topics!

1. Visual and Project-based learning
2. Computing and the Web
3. Programming using Java
4. Simulate Natural Systems
5. Develop Games
6. Learn Math & Physics Concepts
7. Machine Learning (as time permits)
8. AP Exam preparation (optional)

# Course Requirements

1. The CPJava course does not require you to have prior programming experience
2. Prerequisites include Algebra 1
3. For students wanting to take the APCS A exam there will some extra assignments that will need to complete. They will not need to do a Final Project.
4. Students not taking the APCS A will complete a Final Project

# Expectations of Learning

- Advanced Java Programming
- Object Oriented including inheritance
- How to make and maintain your own website
- Some basic HTML & CSS
- How to use GitHub
- Searching and sorting
- Recursion
- Robust code design and style
- APCS A topics
- Code Vectors, Newton's Laws, Machine Learning etc
- + Fun stuff like Asteroids, Super Mario Games and Computer Art that they don't teach you in college!

# CPJava course is online!

- The complete 1-year coursework is online!
- We'll start at the top and work our way to the bottom through this course

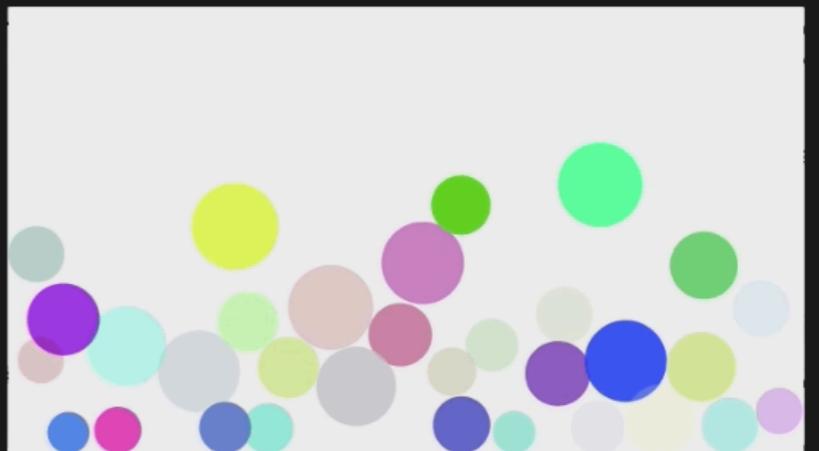
You are here! Click here to access.

Bookmark this  
You will need it every day!



/ CPJava Course  
Bush School Computer Programming in Java Course  
[View on GitHub](#)

*Bush School CPJava Fall Semester 2020*



[Click here for live code and click bubbles inside resulting tab or window](#)

## CPJava - Computer Programming in Java Course

This course is designed to introduce computer programming in the Java language. Learning to use a computer language is a necessary skill for all students regardless of discipline. In this course we will teach the fundamentals of computer programming from the stand point of simulation, automation, and problem solving of real-world systems and natural processes. At the same time, the design and implementation of computer programs is taught from the context of fundamental aspects of computer science, including the development and analysis of algorithms, the development and use of fundamental data structures, the study of standard algorithms and typical applications, and the use of logic and formal methods.

In addition, the year-long course will cover many of the topics necessary for preparation to the AP Computer Science A (APCSA) examination in Spring of the following year. This is an introductory course in computer programming using Java. As such, no specific programming prerequisites are needed to take this course. However, additional preparation may be needed to fully prepare a student for the AP CSA exam with no prior knowledge of computer programming.

# Course Schedule

LESSON UNITS	APPROXIMATE DURATION	TOPICS
<b>FALL 2022 SEMESTER</b>	<b>12-Weeks</b>	<b>Sep 2022 to Jan 2023</b>
Unit 1	2-Weeks	Introduction to Java, Tools walkthrough, Classroom processes, Environment setup, Java Primitive types and Operators
Unit 2	2-Weeks	Objects, Methods, String, Pointers, Integer, Double, Math
Unit 3	2-Weeks	Control flow, Booleans, If statements, Object traversals, Integer, Double, Math
Unit 4	3-Weeks	Iteration, While loops, For loops, Nested Loops, Loop Analysis
Unit 5	3-Weeks	Anatomy of a class, Constructor, Accessor, Mutator Methods, this keyword
<b>SPRING 2023 SEMESTER</b>	<b>15-Weeks</b>	<b>Jan to June 2023</b>
Unit 6	2-Weeks	Arrays, Array Traversal, Data Structures, Project work
Unit 7	3-Weeks	ArrayList, Big data, Ethics of Data Collection, Privacy
Unit 8	2-Weeks	2D Arrays, Traversal, Table representation
Unit 9	3-Weeks	Inheritance, Encapsulation, Hierarchy, Polymorphism, Multi-part Project
Unit 10	3-Weeks	Recursion, Recursive Search, Recursive Sort
Unit 11	2-Weeks	Final Project Peer Sharing Final Project Presentation or APCS A Exam

A complete Syllabus is available at the [CPJava Course website](#)

# Grading Policy

Points are assigned for:

Completing Classwork Exercises

Submitting Projects

Professionalism & Integrity

AP CSA exam is on May 3rd 2023

Talk to me if you are going to take it!

# Grades

Grades are assessed each Semester

50% Projects

(Includes Final Project or APCSA Exam)

35% Classwork / Assignments

15% Student Portfolio

Details are available  
at the CPJava Course website

# How to succeed

Simply write lots of Java code

Publish your work to the web (Github)

You cannot learn programming by reading or  
watching!

Make lots of mistakes - truly the best way to learn  
to code!

Working cooperatively with your peers - Pair  
Programming!

Don't be afraid to try new things!

# Professionalism

- How you conduct yourself during your work
- Includes (but not limited to)
  - Classroom etiquette
  - Reliability and accountability
  - Ethics
  - Working cooperatively with your peers

# Office Hours and Class Assistance

- Conference Hours in Wis 207
  - 3:10 - 3:30 following CPJava Class
  - Extra Conf Hours by Calendly Appt via Portal
- Class Assistance
  - Tully Eva is the TA for this class!

# Tools/Resources for CPJava

CPJava website - Lesson Plans and Projects

Bush Portal - Course Syllabus, Schedule and Grades

CPJava Google Classroom - You should have received an invitation

CPJava Slack - You should have received an invitation

Applications for Laptop

- Github, Processing, Visual Code

- See Instructions in Google Classroom

CSAwesome Online

- Online Textbook, Lessons, Exercises, Detailed Reference

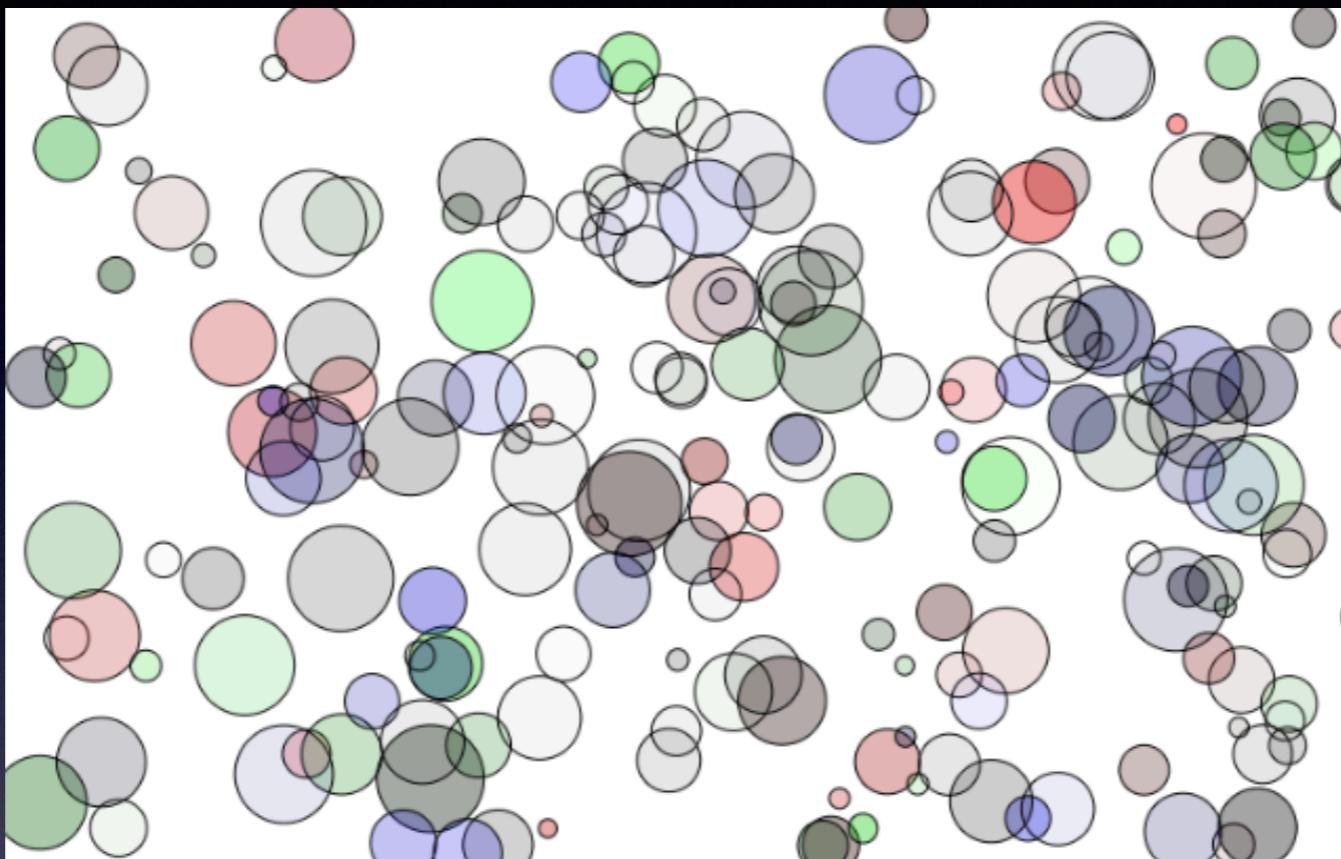
# System Requirements

A laptop Computer - Mac or Windows  
(Chromebook may not be adequate)

Sufficient disk space, a functioning laptop battery  
fully charged!

Functioning Camera and Microphone - especially  
required for Paired Programming

# Let's run our first Java Program!



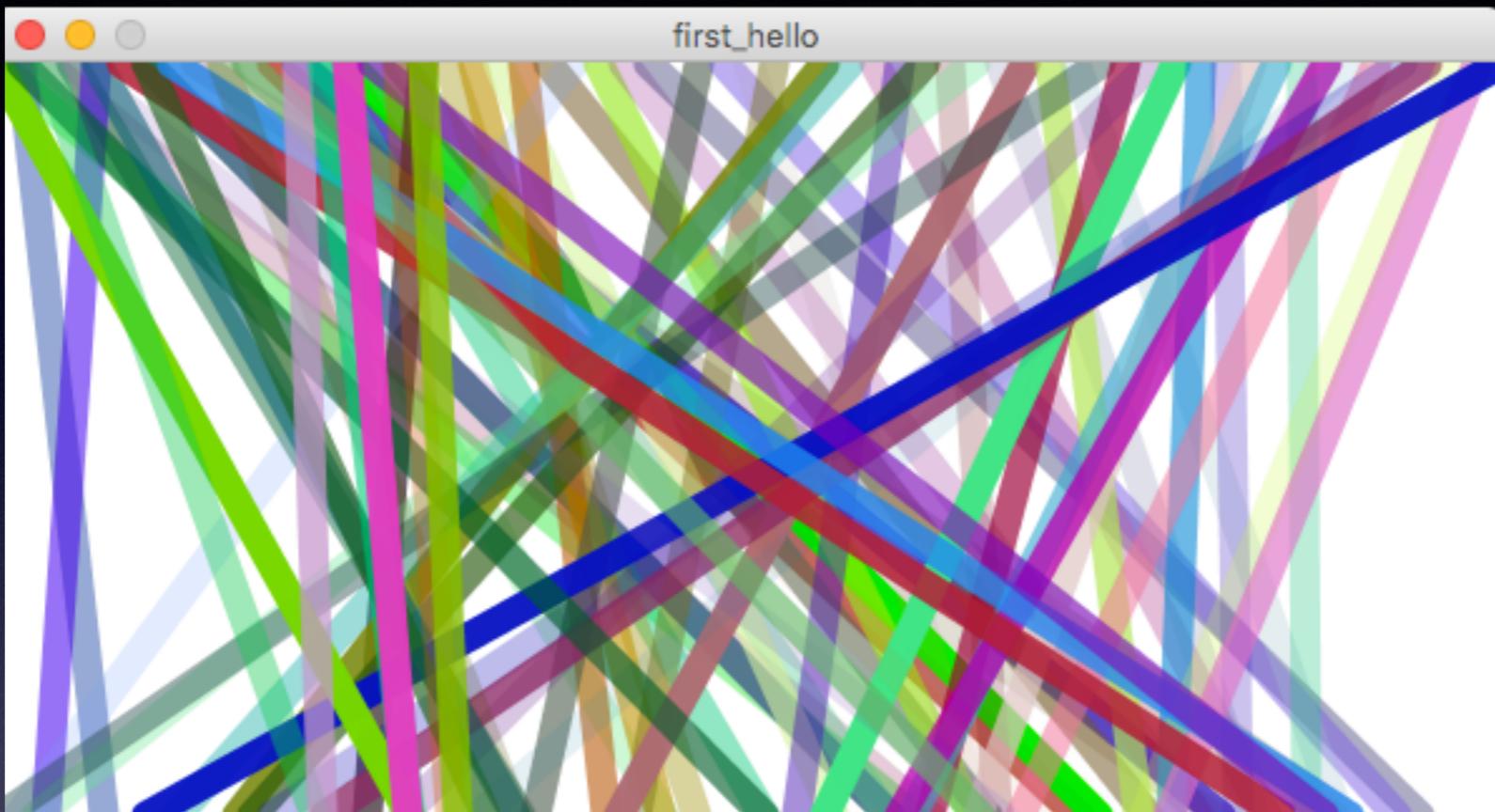
Bubbles - click to run the program

You can sort by color (key 's') and freeze each bubble by clicking on it

Can you freeze all the bubbles?

You will develop visual code like this in Java and much more!

# Let's modify our first Java Program!



Access and complete the First Hello and Sticks Google Classroom Assignment during class!

# Java Basics

## Unit 0

# Unit 0-A

The common building blocks in programming languages  
are:  
Variables  
Loops  
if statements  
Functions (aka “methods”)

Over the next two weeks we’ll go over how these work in  
Java

# A basic Java Hello program

- Click on the link to the “Four 4s Challenge” of just go to CP Java Website
- It might look complicated at first, but . . .



```
1 public class ABasicJavaProgram
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello You!");
6     }
7 }
```

# A basic Java program

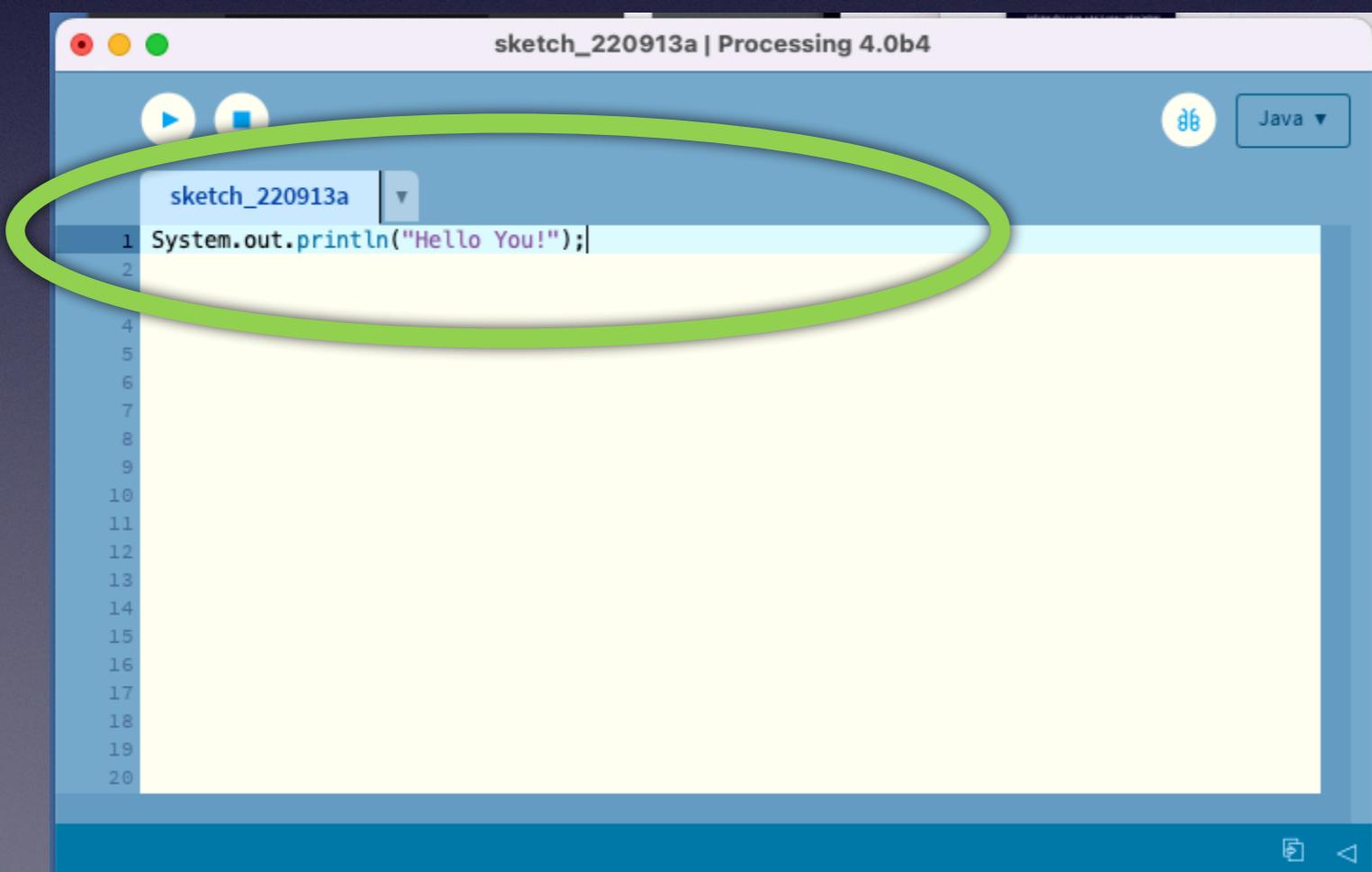
- It turns out that this line is much more important than the others
- So in subsequent slides we can ignore everything except the code statements circled in green - Practice in Processing editor!



```
trinket Java beta Run Share ABasicJavaProgram.java
1 public class ABasicJavaProgram
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello You!");
6     }
7 }
```

# A basic Java program in Processing

- Ignore any references to Trinket in the subsequent slides. Instead we will use Processing. It's Much easier!



The screenshot shows the Processing 4.0b4 IDE window titled "sketch\_220913a | Processing 4.0b4". The code editor displays the following Java code:

```
System.out.println("Hello You!");
```

A large green oval highlights the code area, specifically the line `System.out.println("Hello You!");`. The IDE interface includes a toolbar with play and stop buttons, a status bar at the bottom, and a "Java" dropdown menu in the top right corner.

# Statements

- The circled code is an example of a *statement*
- It's like an English sentence
- A semi-colon marks the end of a Java statement

DO THIS —→



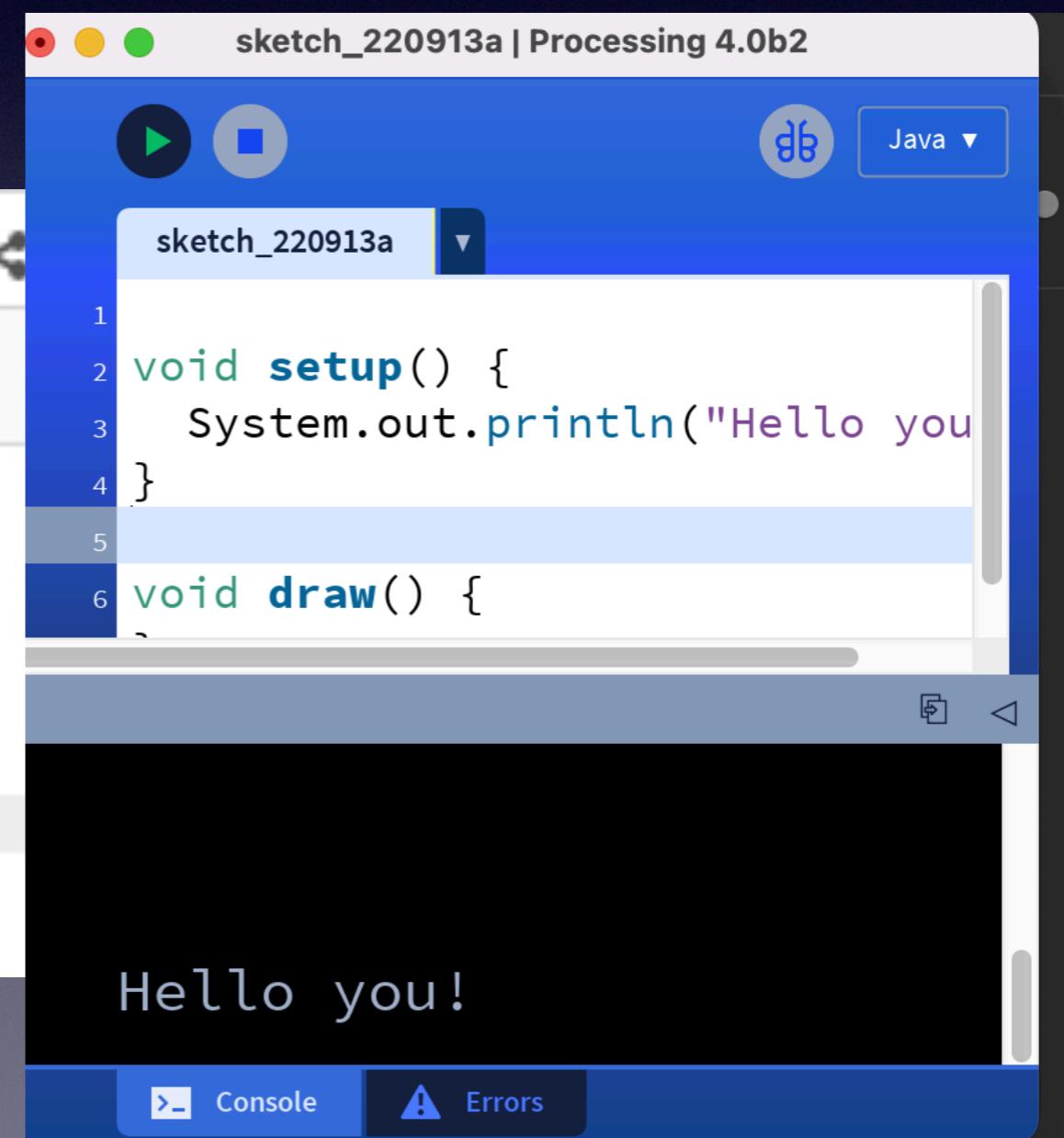
trinket Java beta

ABasicJavaProgram.java

```
1 public class ABasicJavaProgram
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello You!");
6     }
7 }
```

NOT THIS

A green oval highlights the line `System.out.println("Hello You!");`. A yellow arrow points from the word "THIS" to this highlighted line.



sketch\_220913a | Processing 4.0b2

sketch\_220913a

```
1 void setup() {
2     System.out.println("Hello you")
3 }
4
5 void draw() {
6 }
```

Hello you!

Console Errors

# String

- "Hello You!" is a Java **String**
- A **String** is a collection of letters, digits, punctuation and/or spaces
- The beginning and end of the **String** are marked with double quotes ("")



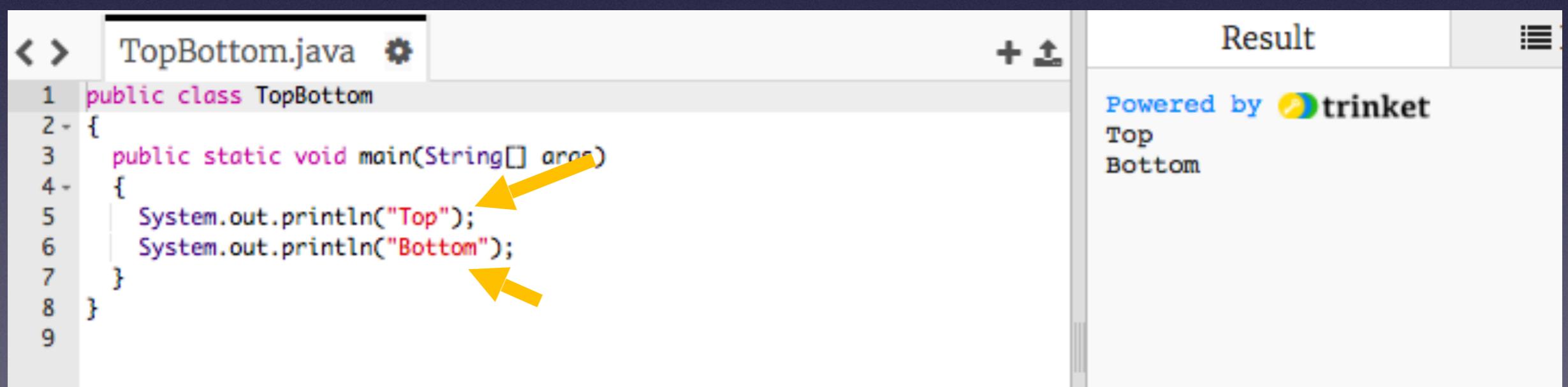
The screenshot shows a Java code editor interface from trinket. The title bar says "trinket Java beta". The file name is "ABasicJavaProgram.java". The code is:

```
1 public class ABasicJavaProgram
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello You!");
6     }
7 }
```

A green oval highlights the string literal "Hello You!" in line 5.

# print() vs println()

- `System.out.println()` prints first and then goes to the next line



The screenshot shows a Java code editor and a results panel from the trinket platform.

**Code:**

```
1 public class TopBottom
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Top");
6         System.out.println("Bottom");
7     }
8 }
```

**Result:**

Powered by trinket

Top  
Bottom

Two yellow arrows point to the two `System.out.println` statements in the code editor, highlighting the behavior described in the slide.

# print() vs println()

- `System.out.print()` prints, but it does NOT go to the next line
- If we change the first statement to `System.out.print()` "Bottom" is printed on the same line as "Top"



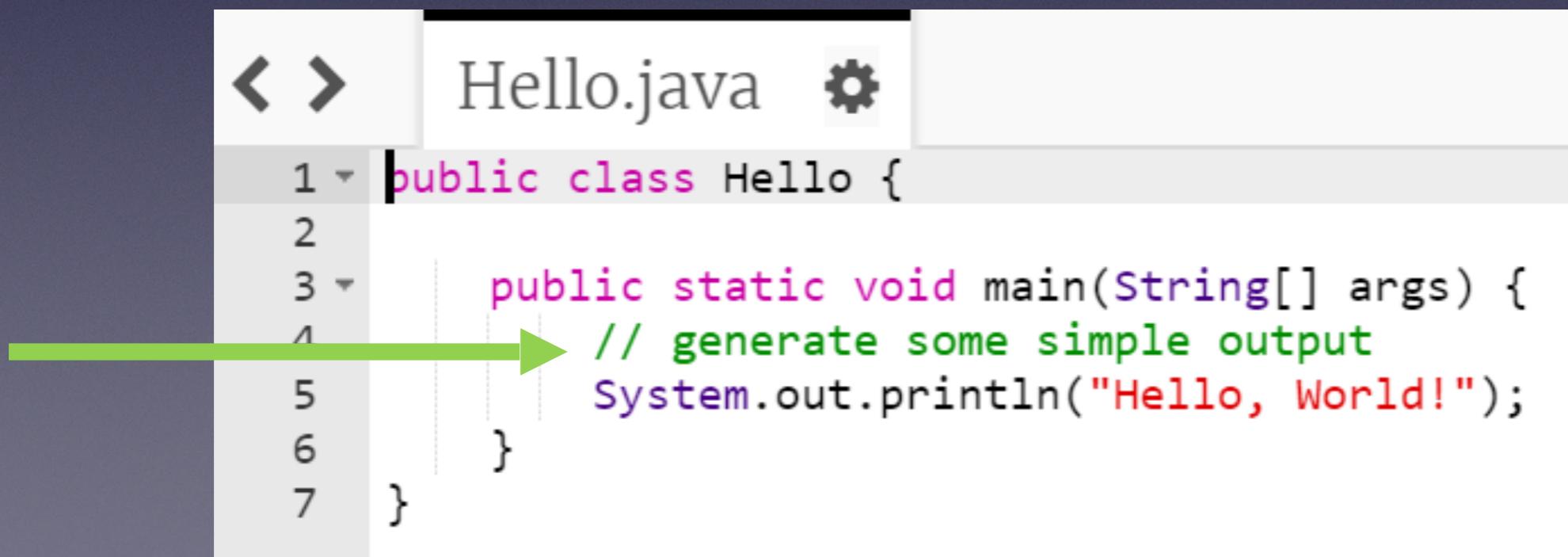
The screenshot shows a Java code editor interface. The top bar includes icons for file, run, and share, along with tabs for Java beta and Run. The main window displays a file named Goodbye.java with the following code:

```
1 public class Goodbye {  
2  
3     public static void main(String[] args) {  
4         System.out.print("Top");  
5         System.out.println("Bottom");  
6     }  
7 }
```

A yellow arrow points to the `System.out.print("Top");` line. To the right of the code editor, there is a sidebar with the text "Powered by trinket" and "TopBottom" followed by a green horizontal bar.

# Comments

- Comments have no effect on the execution of the program, but they make it easier for other programmers (and your future self) to understand what you meant to do



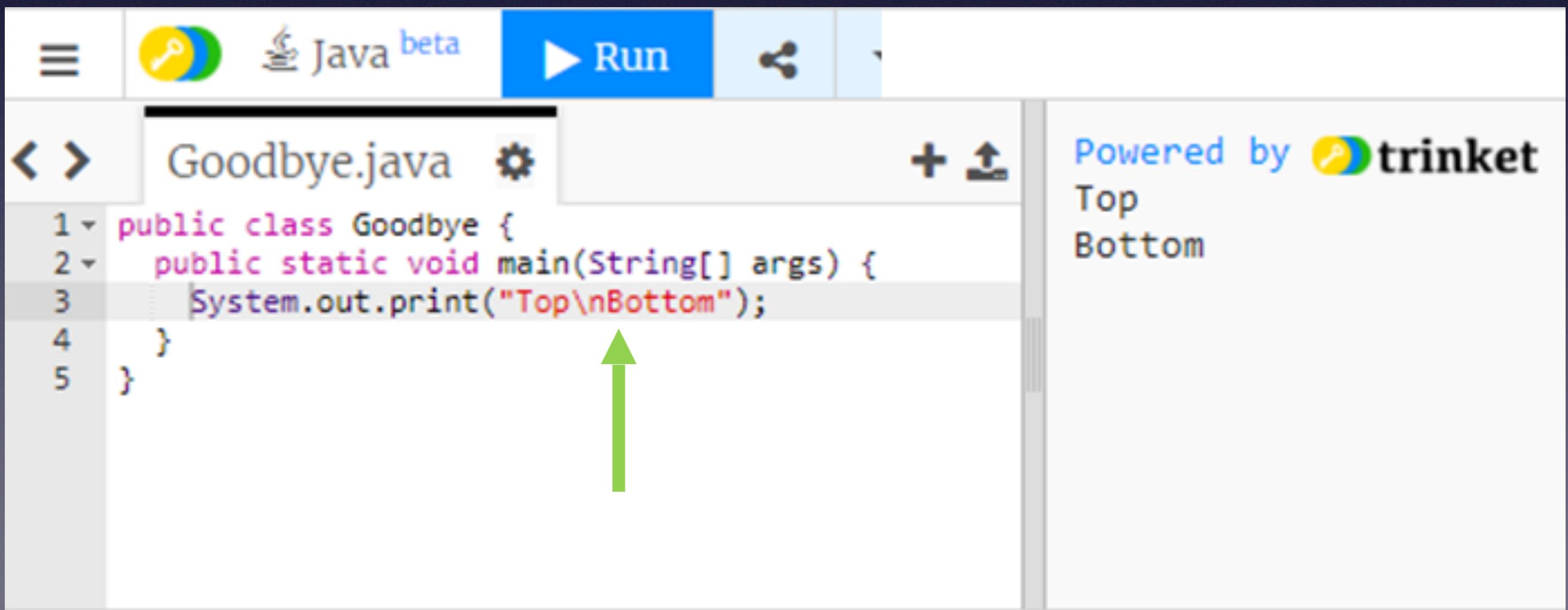
```
1 < > public class Hello {  
2  
3     public static void main(String[] args) {  
4         // generate some simple output  
5         System.out.println("Hello, World!");  
6     }  
7 }
```

# Escape Sequences

- Special characters
- In Java, Escape Sequences begin with a backslash \
- *A good way to remember the difference between a backslash and a forward slash is that a backslash leans backwards ( \ ), while a forward slash leans forward ( / )*

# Common Java Escape Sequences

- \n Insert a newline in the text at this point



```
Goodbye.java
public class Goodbye {
    public static void main(String[] args) {
        System.out.print("Top\nBottom");
    }
}
```

Powered by  trinket  
Top  
Bottom

# Common Java Escape Sequences

- `\t` Insert a tab in the text at this point

The screenshot shows a Java code editor interface with the following details:

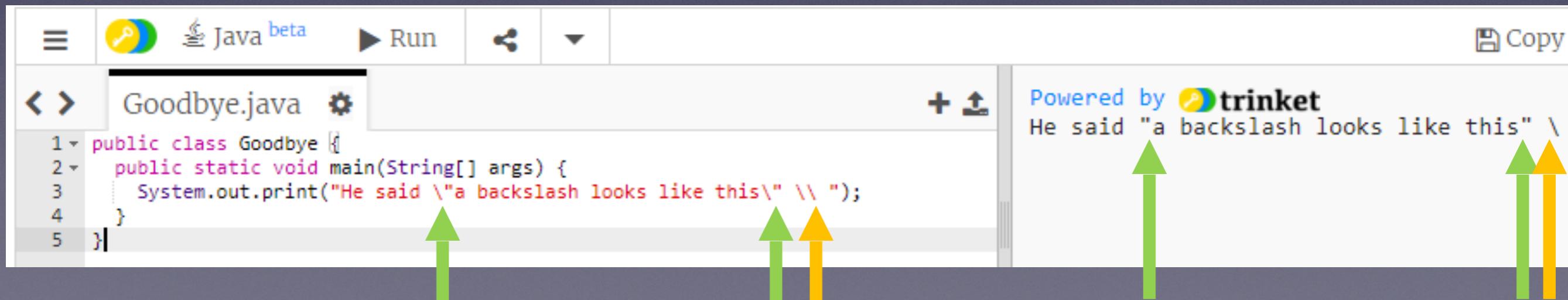
- Toolbar:** Includes icons for file operations, a key icon, "Java beta", "Run", and other development tools.
- File List:** Shows "Goodbye.java" as the active file.
- Code Editor:** Displays the following Java code:

```
1 public class Goodbye {  
2     public static void main(String[] args) {  
3         System.out.print("x\tx\tx\tx");  
4     }  
5 }
```

Three green arrows point upwards from the bottom of the slide towards the three tabs in the string "System.out.print("x\tx\tx\tx");".
- Output Area:** Shows the output "x x x x" followed by four green double-headed arrows and the text "Powered by trinket".

# Common Java Escape Sequences

- `\'` Insert a single quote character
- `\\"` Insert a double quote character
- `\\"\\` Insert a backslash character



A screenshot of a Java code editor showing a file named `Goodbye.java`. The code contains the following:

```
1 public class Goodbye {  
2     public static void main(String[] args) {  
3         System.out.print("He said \"a backslash looks like this\" \\\\ ");  
4     }  
5 }
```

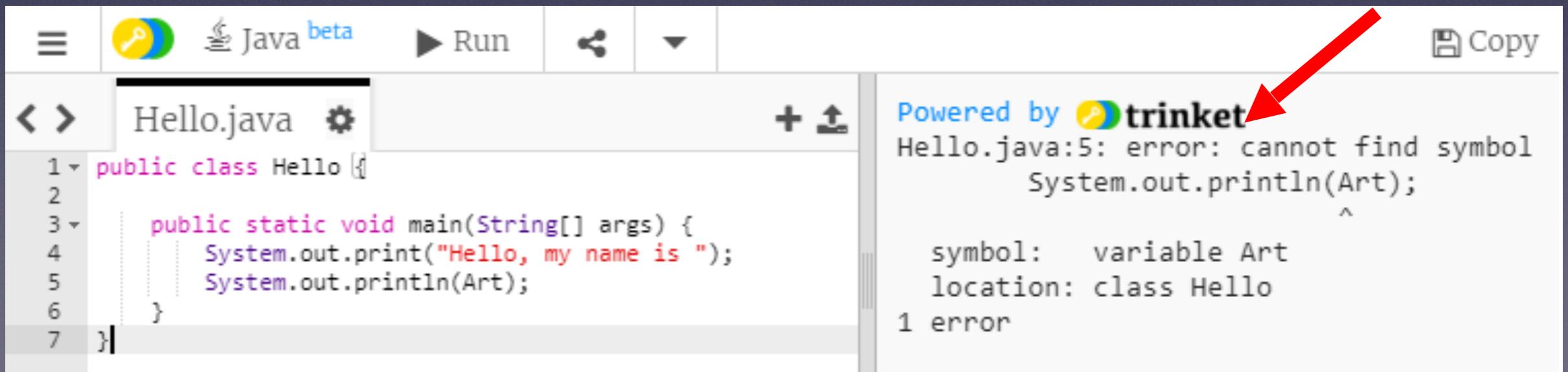
The output window shows the result of the `System.out.print` statement: "He said "a backslash looks like this"" followed by a backslash character. Three green arrows point from the escape sequences in the code to the corresponding characters in the output. One green arrow points to the double quotes in the string, another points to the backslash before the final closing double quote, and a third points to the backslash character itself.

# Debugging

- Errors in programs are called “bugs”
- The process of fixing program errors is called “debugging”
- It’s good to work around other programmers when you are learning a new programming language
- Asking for help with debugging is a part of learning

# Errors

- When you write Java programs you will often get an **error message**
- When you are learning a new programming language, errors are a fact of life
- Errors are ok, just fix them and move on



The screenshot shows a Java code editor interface. The top bar includes icons for file operations, a key icon, "Java beta", a run button, and a copy button. The main area displays a Java file named "Hello.java" with the following code:

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         System.out.print("Hello, my name is ");  
4         System.out.println(Art);  
5     }  
6 }  
7 }
```

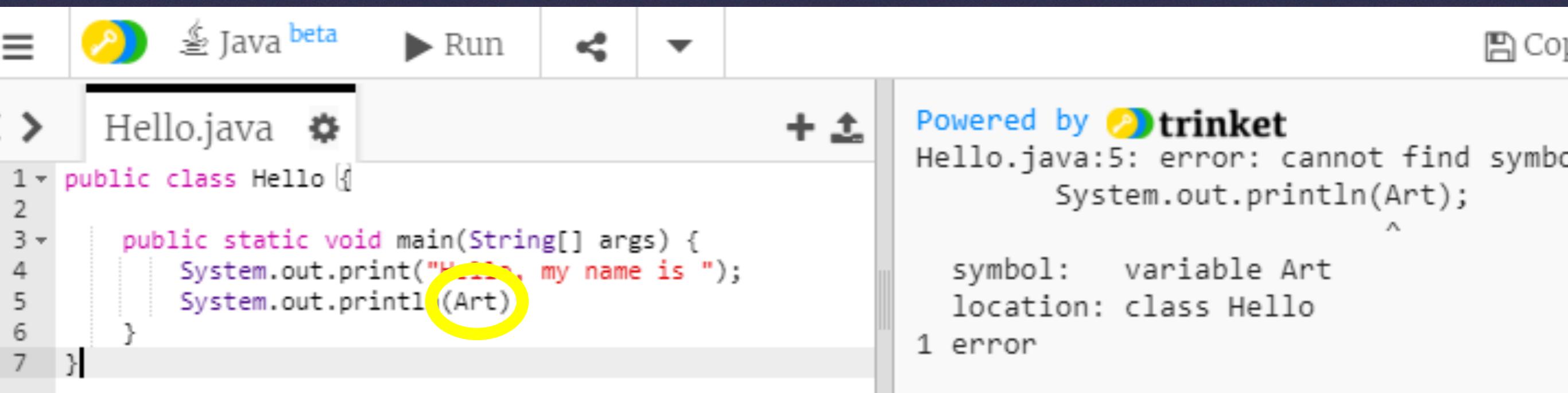
To the right of the code, the status bar shows the output of the Java compiler:

Powered by  trinket  
Hello.java:5: error: cannot find symbol  
System.out.println(Art);  
^  
symbol: variable Art  
location: class Hello  
1 error

A red arrow points from the bottom right towards the "Powered by trinket" text.

# Syntax error

- In this case I made a *syntax* error
- *Syntax* is the grammar and spelling of a computer language
- Here I forgot the double quotes around my name



The screenshot shows a Java code editor interface. The top bar includes icons for file, Java beta, run, share, and copy. The code editor window displays a file named "Hello.java". The code is as follows:

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         System.out.print("Hello, my name is ");  
4         System.out.println(Art);  
5     }  
6 }  
7 }
```

A yellow circle highlights the word "Art" in the line "System.out.println(Art);". The status bar on the right indicates the error: "Powered by trinket" and "Hello.java:5: error: cannot find symbol System.out.println(Art); ^ symbol: variable Art location: class Hello 1 error".

# Logic error

- This time I misspelled my name
- The computer doesn't know my name, so the program runs incorrectly without an error message



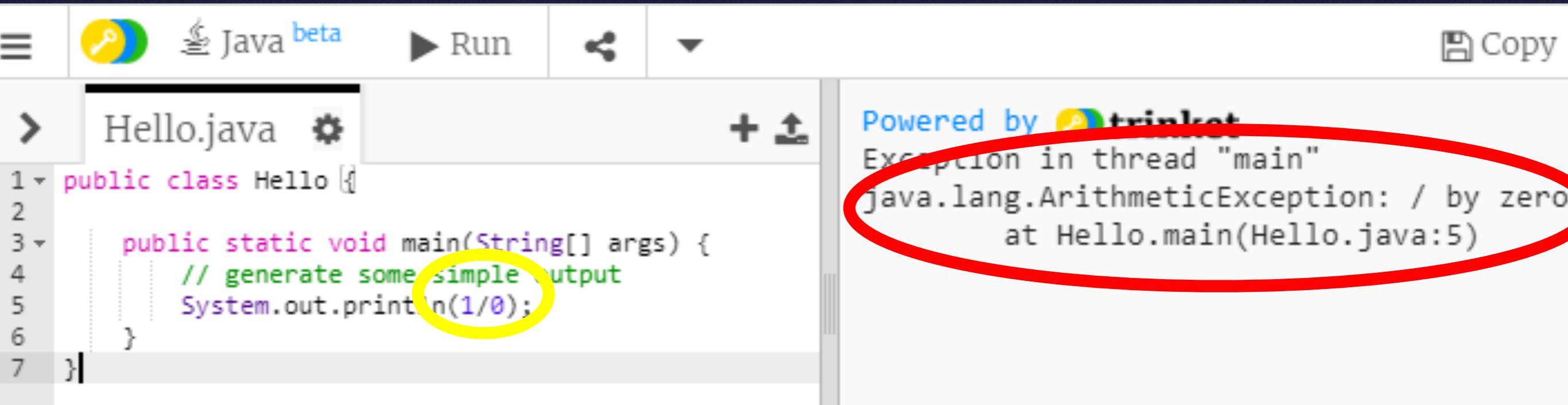
The screenshot shows a Java code editor interface. The top bar includes a logo, the text "Java beta", and a "Run" button. The code editor window displays a file named "Hello.java". The code contains a simple "Hello, world!" program with a misspelling:

```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         System.out.print("Hello, my name is ");  
5         System.out.println("Arg");  
6     }  
7 }
```

A yellow oval highlights the misspelling "Arg" in the println statement. To the right, the output window shows the result of running the program: "Hello, my name is Arg". The word "Arg" is circled in green, indicating it is the error being discussed.

# (Run time) Exceptions

- Sometimes a logic error crashes the computer and stops the running program
- Here I made the logic error of dividing by zero



```
Java beta
```

Run

Copy

Hello.java

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         // generate some simple output  
4         System.out.println(1/0);  
5     }  
6 }  
7 }
```

Powered by trinket

Exception in thread "main"  
java.lang.ArithmetricException: / by zero  
at Hello.main(Hello.java:5)

# Arithmetc in Java

**+**   **-**   **\***   **/**

- Addition
- Subtraction
- Multiplication
- Division

# Literals vs. Expressions

- Double quotes around text tells Java it is an expression
- Java will **print** an expression exactly as written

# Literals vs. Expressions

- Here's an expression "4/4"
- Java prints it in exactly the same form

The screenshot shows a Java development environment with the following details:

- Toolbar:** Includes icons for file operations, a key icon, Java beta logo, Run button, and share/icon buttons.
- Project Explorer:** Shows "Hello.java" selected.
- Code Editor:** Displays the following Java code:

```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         System.out.println("4/4");  
5         System.out.println(4/4);  
6     }  
7 }
```

The line `System.out.println("4/4");` is circled in red.
- Output Window:** Shows the output "Powered 4/4" with the "4/4" part circled in red.

# Literals vs. Expressions

- Here's an expression  $4 / 4$
- Java evaluates it to get an answer 1
- And then prints it

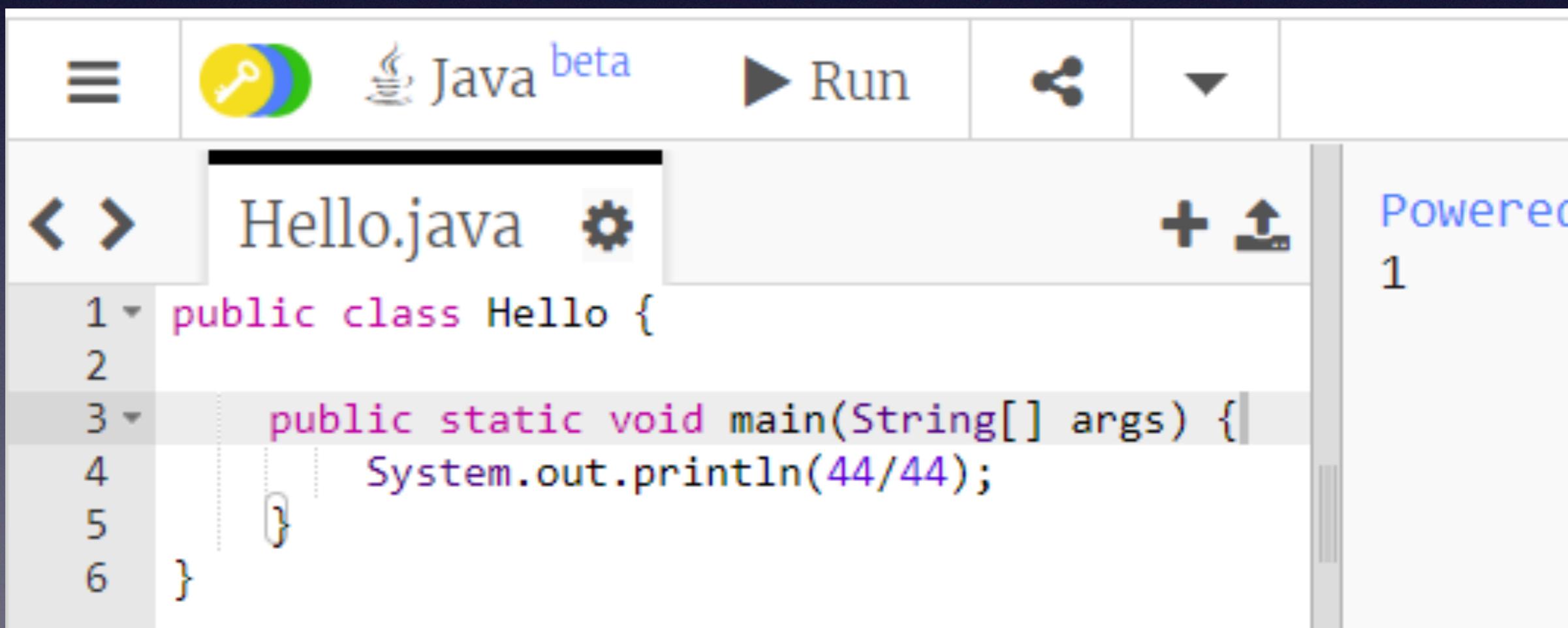
```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("4/4");  
        System.out.println(4/4);  
    }  
}
```

# Four 4s challenge

- Use exactly four 4's to write an expression that evaluates to every integer from 1 to 10, using only  $+$   $-$   $*$   $/$  and  $( )$
- No decimals, factorials, square roots, exponents, etc.

# Four 4s challenge

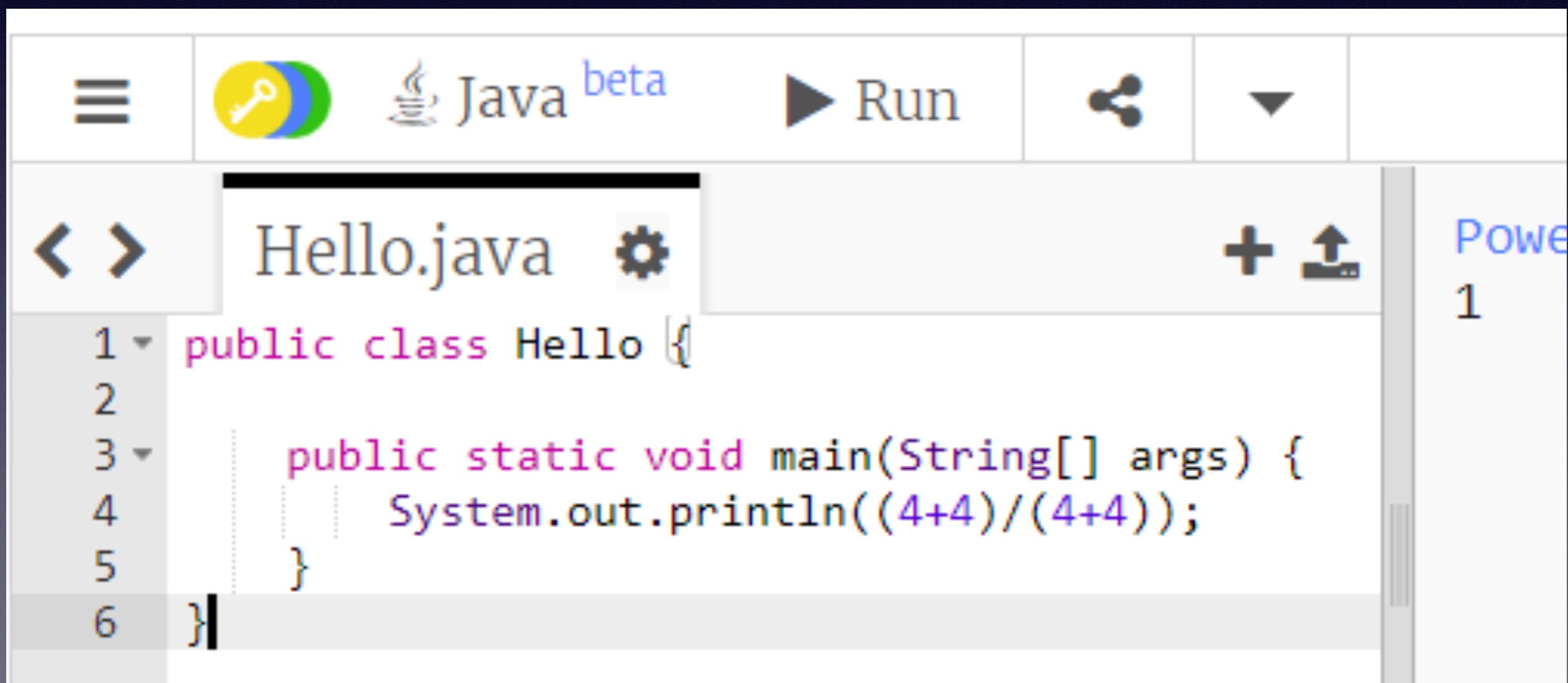
- Print 10 expressions that use arithmetic and four 4s that evaluate to 1 through 10
- Here's one way to do the first



```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println(44/44);  
    }  
}
```

# Four 4s challenge

- Here's another way to do the first
- If you have extra time, try to get 11, 12, 13, etc.



The screenshot shows a Java code editor interface. The top bar includes a key icon, the text "Java beta", a "Run" button, and other standard icons. Below the bar, the file "Hello.java" is selected. The code editor displays the following Java code:

```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         System.out.println((4+4)/(4+4));  
5     }  
6 }
```

The code contains a syntax error: the closing brace for the main method is placed on the same line as the opening brace, which is invalid Java syntax. This is likely the "bug" mentioned in the slide title.

# Unit 0-B

# Java Functions and Parameters

# Functions (“methods”)

- Organize code just like paragraphs organize writing
- Functions should do just one job or task
- Functions in Java are identified with parenthesis
- The parenthesis may or **may not** have something inside called an **argument**

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

```
in.nextInt();
```

no argument



argument

# Functions can be used for an *effect* and/or *value*

- `System.out.println()` has an *effect*, it causes something to be displayed
- `in.nextInt()` is used to get the *value* of what the user typed
- Other functions that we will learn about in a couple days can be used to calculate mathematical *values*

# void functions (aka methods)

- You can *define* (create) your own functions that are used for effect with the Java keyword **void**
- Practice in Processing ignoring only the crossed out items! See next 2 slides**

NewLine.java

```
1 public class NewLine {  
2  
3     public static void blah() {  
4         System.out.println("blah, blah, blah");  
5     }  
6  
7     public static void main(String[] args) {  
8         blah();  
9     }  
10}
```

Powered by trinket  
blah, blah, blah

# void functions (aka methods)

- It should look like this !
- Follow same pattern for subsequent slides

The screenshot shows the Processing IDE interface with the title bar "sketch\_220913a | Processing 4.0b4". The code editor contains the following Java code:

```
void blah() {
    System.out.println("blah, blah, blah");
}

void setup() {
    blah();
}

void draw() {
```

Annotations with red arrows and text boxes explain the code:

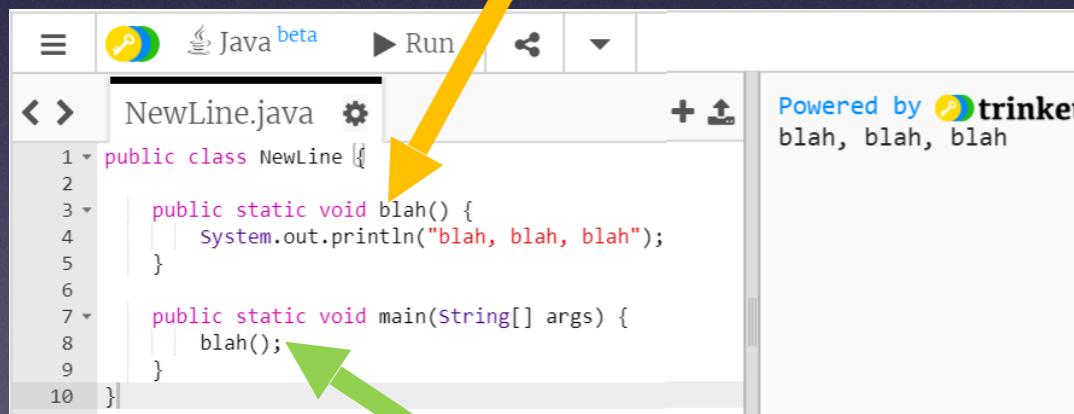
- A red box surrounds the `void blah()` function definition. A red arrow points from this box to the text: "This is a function or method".
- A red arrow points from the `blah()` call in the `setup()` function to the text: "This setup() function calls the blah() function".
- A red arrow points from the empty `draw()` function body to the text: "The draw() function is doing nothing now".
- The text: "What is you moved the call for the blah() function from the setup() function() to the draw() function ??"

The status bar at the bottom of the IDE shows the text "blah, blah, blah".

# void functions (aka methods)

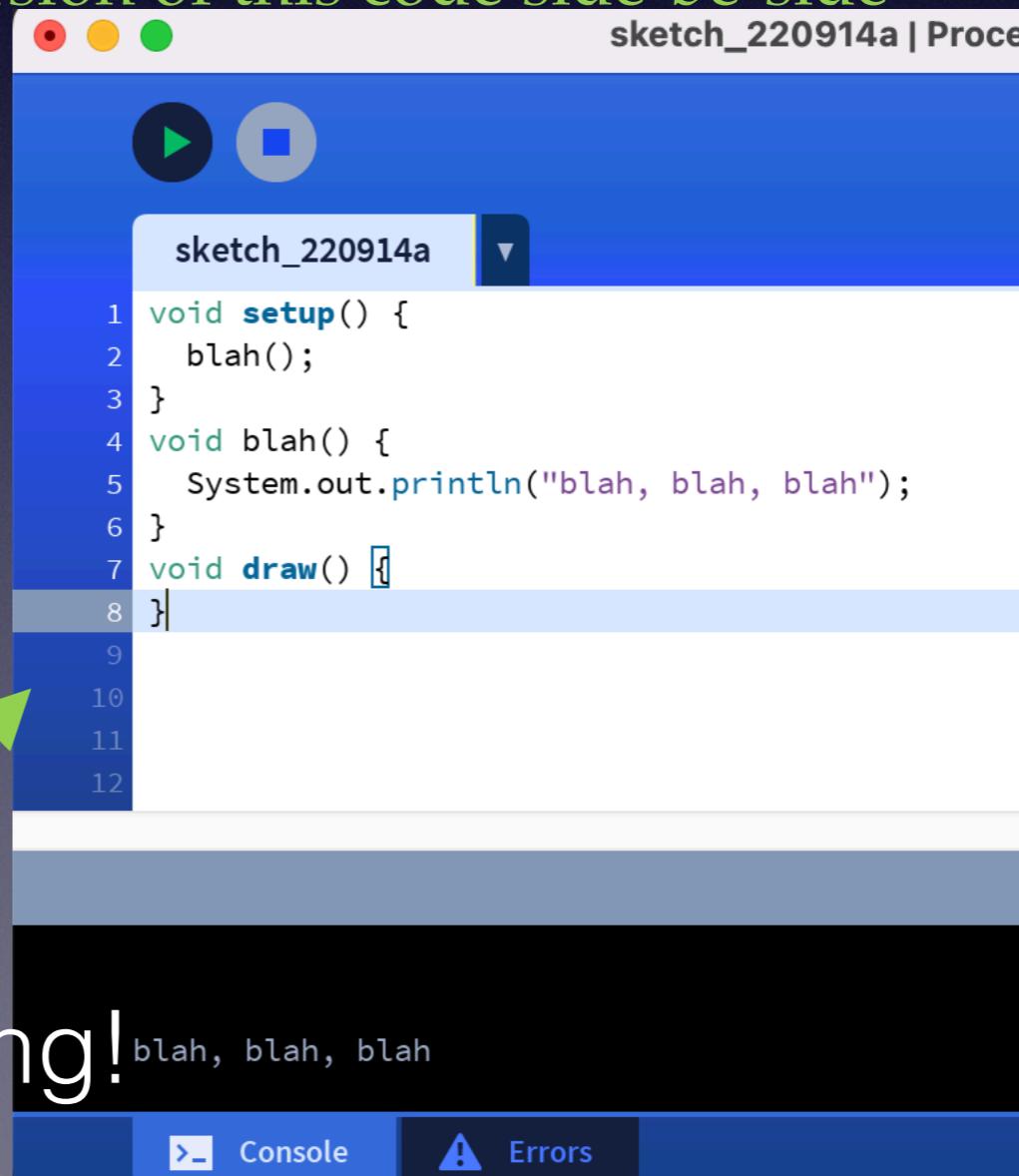
- Functions are like paragraphs of computer code
- Every Java program has a **main** function. There can be many other functions
- More on this later. For now we will use processing which does not need the main() function. Processing version of this code side-be-side

Function blah()



```
1 public class NewLine {  
2     public static void blah() {  
3         System.out.println("blah, blah, blah");  
4     }  
5     public static void main(String[] args) {  
6         blah();  
7     }  
8 }
```

main()



```
1 void setup() {  
2     blah();  
3 }  
4 void blah() {  
5     System.out.println("blah, blah, blah");  
6 }  
7 void draw() {}  
8 
```

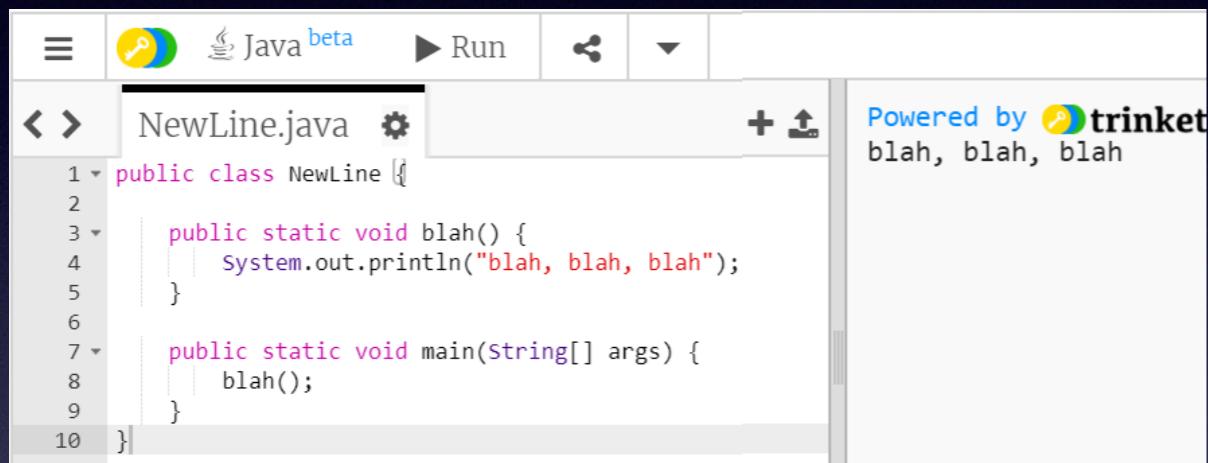
blah, blah, blah

Console Errors

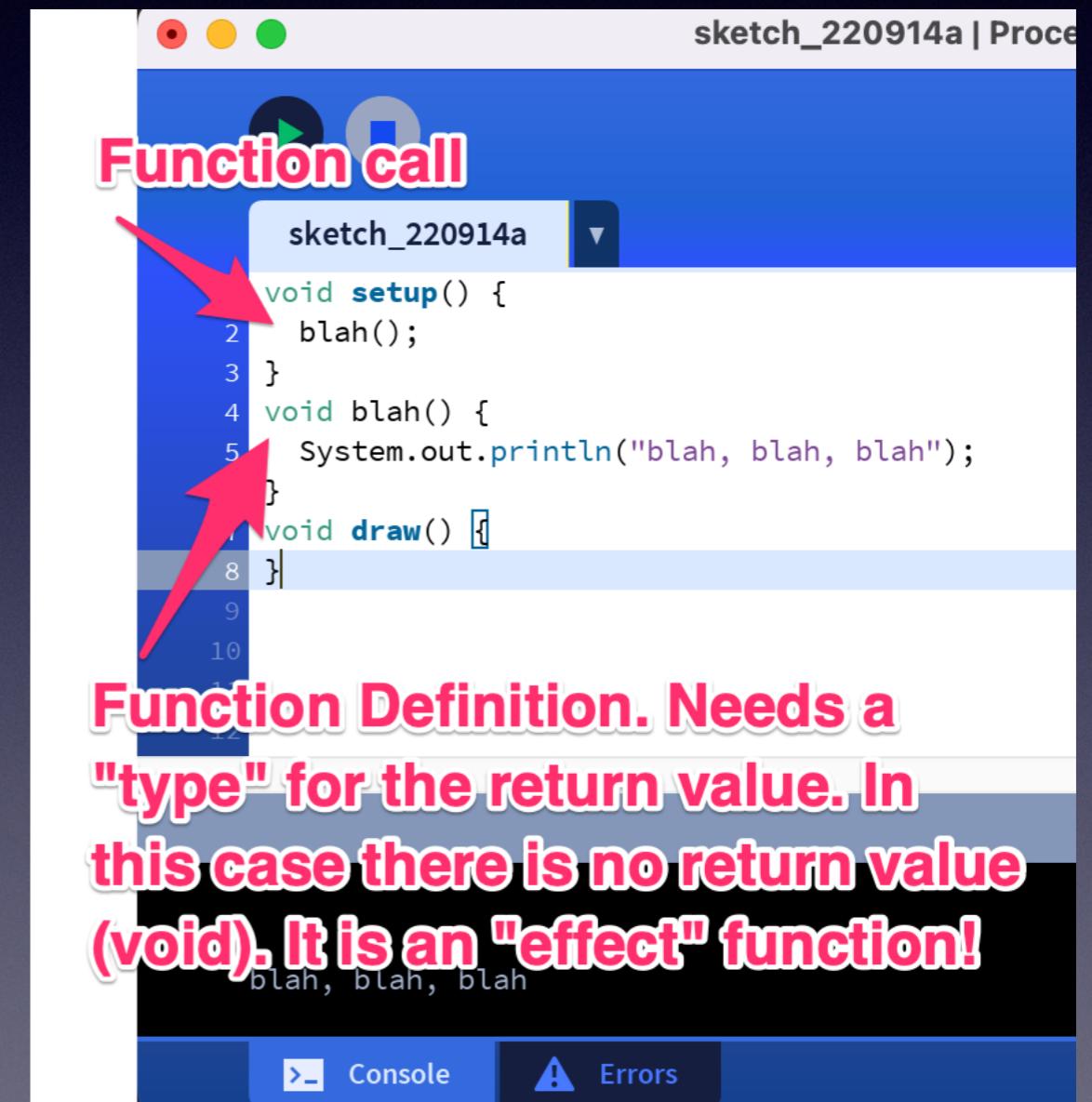
No main() needed in processing!

# void functions (aka methods)

- You can *call* (use) your function by typing its name without **void**



```
1 public class NewLine {  
2     public static void blah() {  
3         System.out.println("blah, blah, blah");  
4     }  
5     public static void main(String[] args) {  
6         blah();  
7     }  
8 }
```



Function call

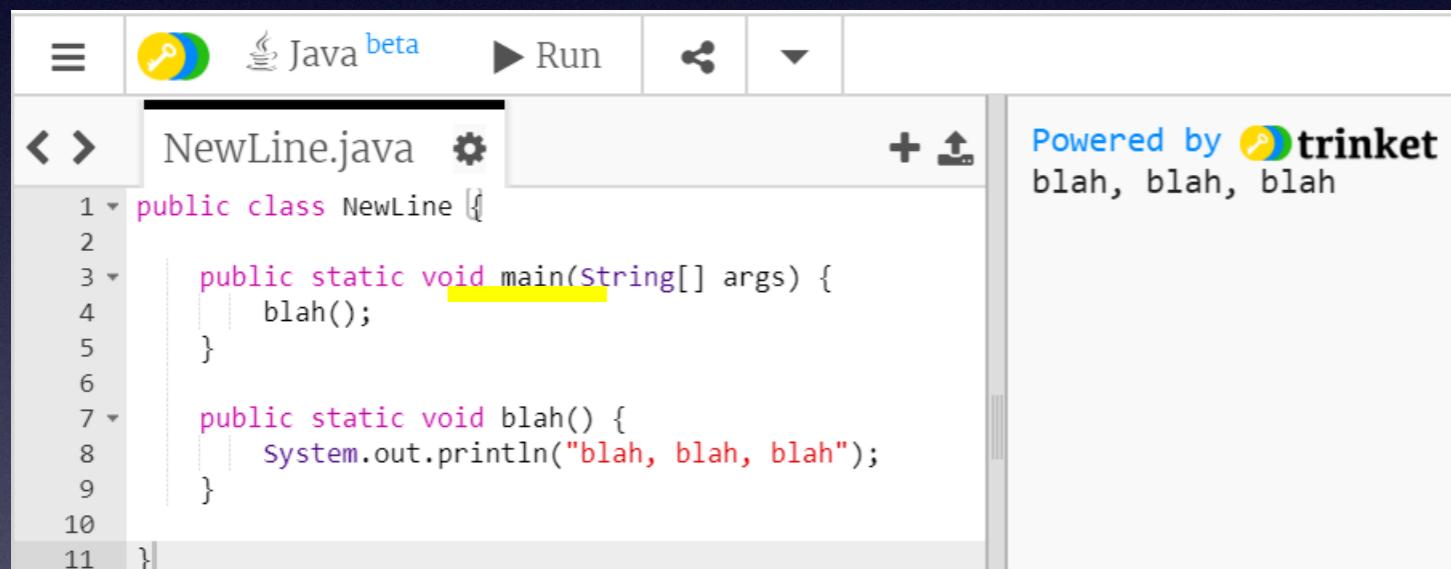
```
sketch_220914a  
void setup() {  
  blah();  
}  
void blah() {  
  System.out.println("blah, blah, blah");  
}  
void draw() {}
```

Function Definition. Needs a "type" for the return value. In this case there is no return value (**void**). It is an "effect" function!

Console Errors

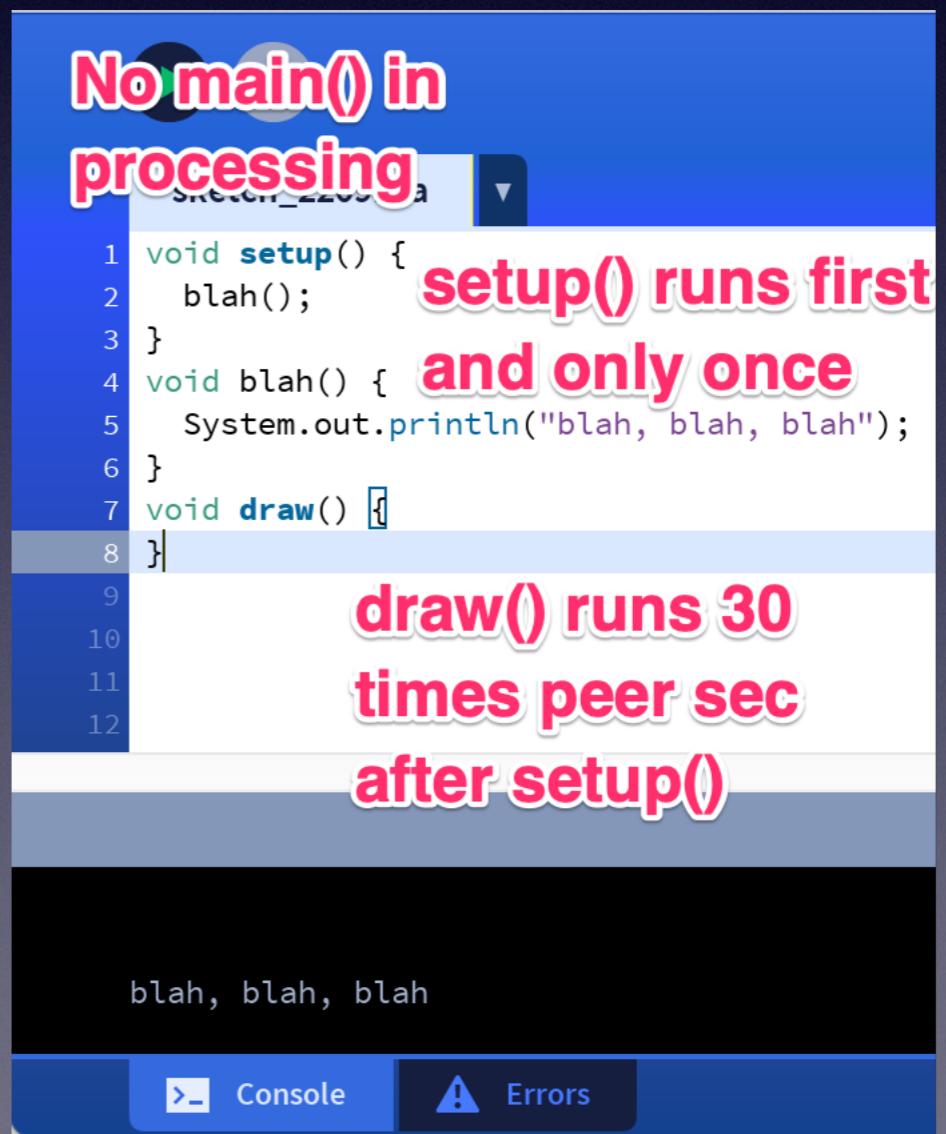
# void functions (aka methods)

- Note that the order the functions are *defined* is unimportant
- Java always runs the **main** function first



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

    public static void blah() {
        System.out.println("blah, blah, blah");
    }
}
```



No main() in processing

```
void setup() {
    blah();
}
void blah() {
    System.out.println("blah, blah, blah");
}
void draw() {
```

setup() runs first  
and only once

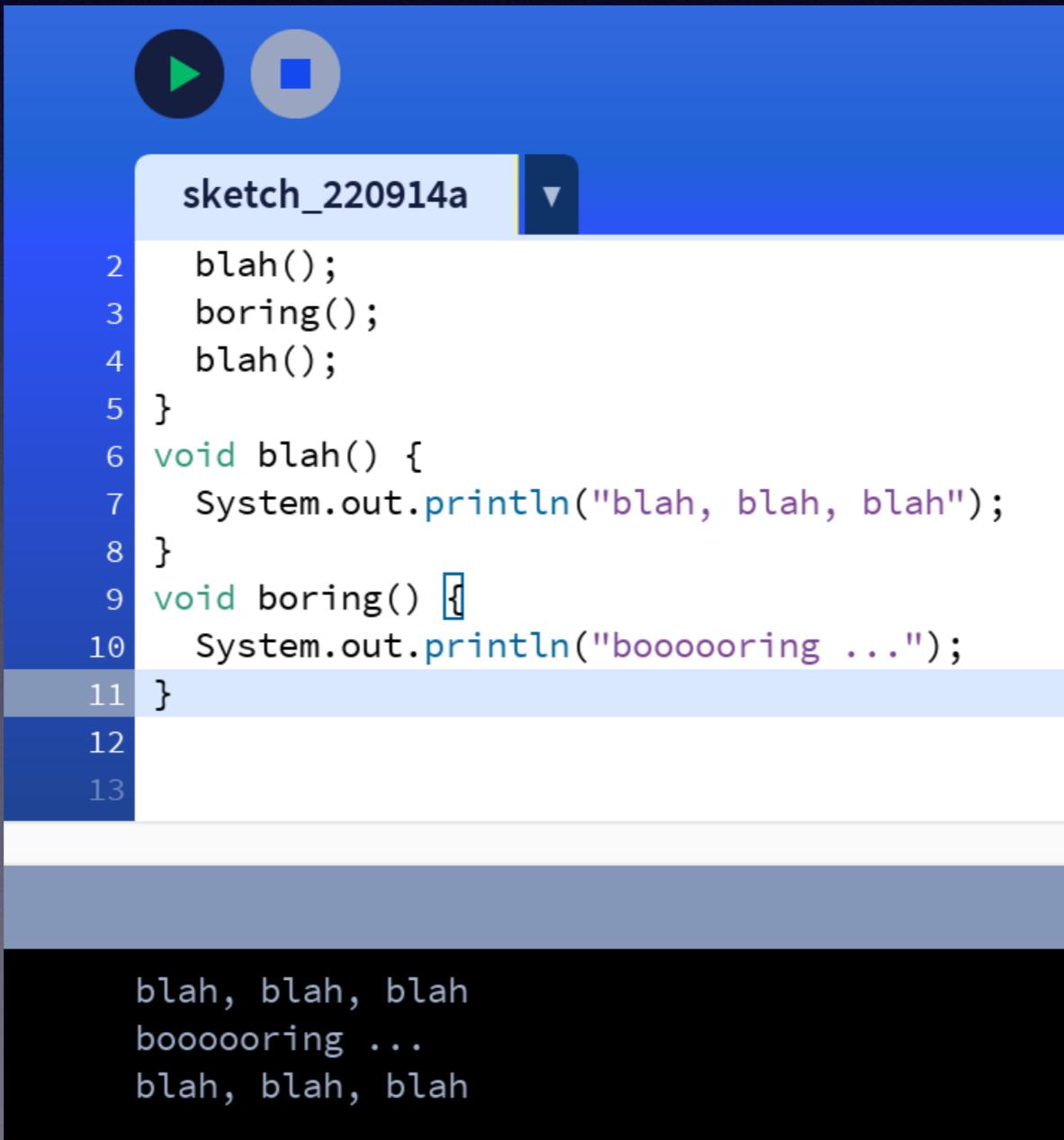
draw() runs 30  
times per sec  
after setup()

blah, blah, blah

Console Errors

# void functions (aka methods)

- The order of the function *calls* is important
- Functions can be reused as many times as you want



The screenshot shows the Processing IDE interface. At the top, there are play and stop buttons. Below them, the sketch name is "sketch\_220914a". The code area contains the following:

```
2 blah();
3 boring();
4 blah();
5 }
6 void blah() {
7     System.out.println("blah, blah, blah");
8 }
9 void boring() {
10    System.out.println("boooooring ...");
11 }
12
13
```

The output window at the bottom displays the results of running the sketch:

```
blah, blah, blah
boooooring ...
blah, blah, blah
```

- Variable declarations in the parenthesis of the function definition are called *parameters*
- The function call has matching *arguments*
- The values in variables **h** & **m** are copied into variables **hour** & **minute**

The screenshot shows a Java code editor window titled "print\_time | Pr". The code defines a function named `printTime` that prints the hour and minute. It also contains `setup` and `draw` methods that call `printTime`. The variable declarations `int h = 7;` and `int m = 35;` are highlighted with a blue box. Two arrows point from these declarations to the corresponding arguments in the `printTime` call: a red arrow points from `h` to `hour`, and a green arrow points from `m` to `minute`.

```
1 int h = 7;
2 int m = 35;
3
4 void printTime(int hour, int minute) {
5     System.out.print(hour);
6     System.out.print(":");
7     System.out.print(minute);
8 }
9
10 void setup() {
11     printTime(h, m);
12 }
13 void draw()
```

7:35

# Arguments must match Parameters in number, order and type

- What's the problem?
- Order, the parameters are **int String** order but the arguments are in **String int** order

The screenshot shows a code editor with a blue header bar containing play and stop buttons. Below the header, the code is displayed:

```
some_function
1 void someFunction(int num, String word) {
2     System.out.println(num + ", " + word);
3 }
4
5 void setup() {
6     int n = 1;
7     String w = "one";
8     someFunction(w, n);
9 }
10
11 void draw() {
12 }
```

Annotations are present in the code:

- A red arrow points from the `someFunction` function name at the top to the `w` parameter in line 8.
- A blue arrow points from the `someFunction` function name at the top to the `n` parameter in line 8.
- A blue arrow points from the `someFunction` call in line 8 up to the `someFunction` definition at the top.
- A red arrow points from the `someFunction` call in line 8 down to the `someFunction` definition at the top.

At the bottom of the editor, a red bar contains the text: "The function 'someFunction()' expects parameters like: 'someFunction(int, String)'".

# Variables can only be used int the function where they are declared

- What's the problem?
- **n** and **w** were **declared** in the **main** function and are not available in **someFunction**

The screenshot shows the Processing 4.0b2 interface with the following code:

```
some_function
1 void someFunction(int num, String word) {
2     System.out.println(n + ", " + w);
3 }
4
5 void setup() {
6     int n = 1;
7     String w = "one";
8     someFunction(n, w);
9 }
10
11 void draw() {
12 }
```

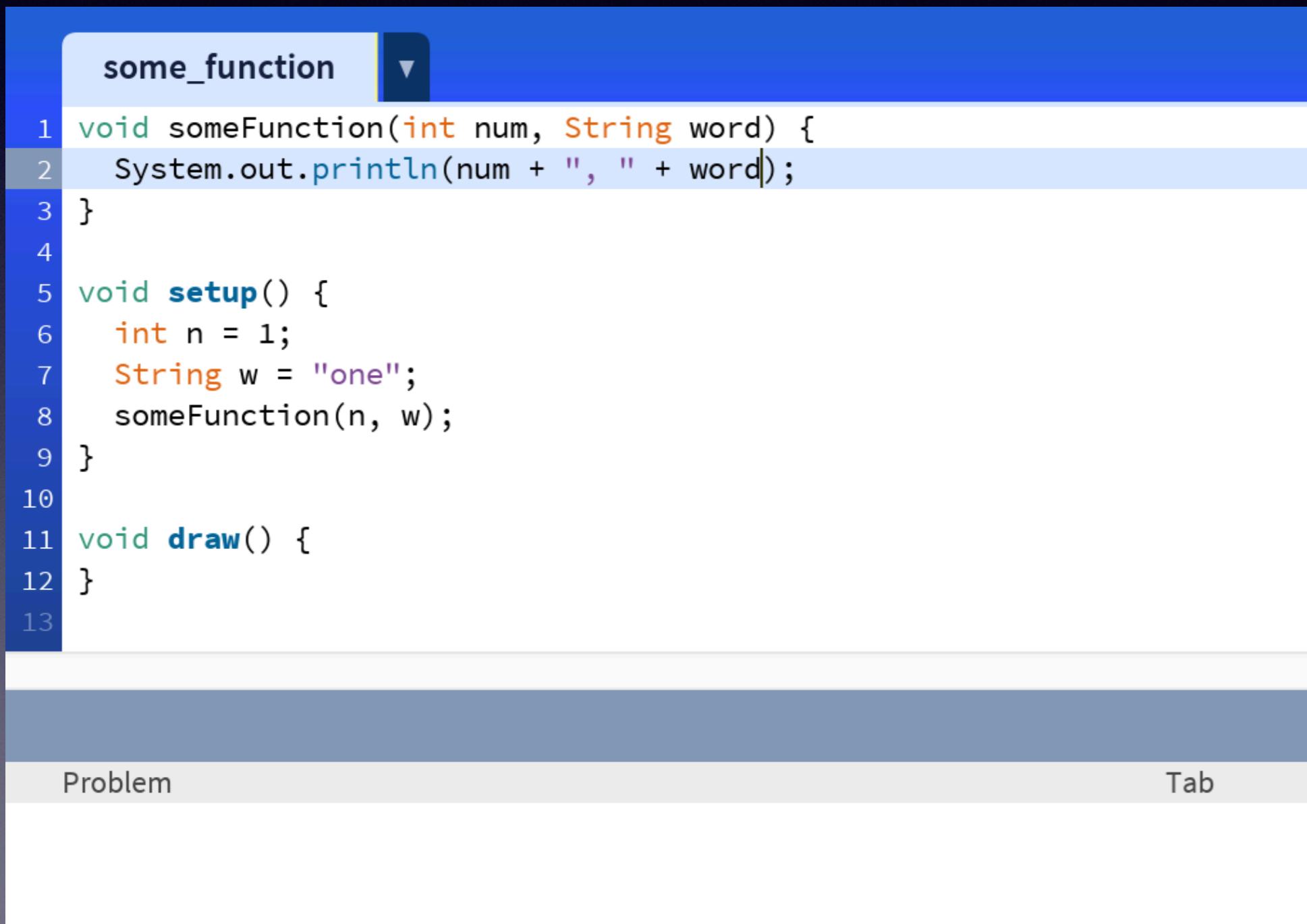
A red error bar at the bottom highlights the line `System.out.println(n + ", " + w);` with the message: "The variable "n" does not exist". The error details table shows two entries:

Problem	Tab	Line
• The variable "n" does not exist	some_function	2
• The variable "w" does not exist	some_function	2

At the bottom, there are "Console" and "Errors" tabs, and a status bar showing "1:35".

# Now the program is correct

- The arguments match the parameters in number order and type
- The correct variables are used in each function



The screenshot shows a code editor window with a dark theme. At the top, there is a tab labeled "some\_function". The code itself is a Java program:

```
1 void someFunction(int num, String word) {  
2     System.out.println(num + ", " + word);  
3 }  
4  
5 void setup() {  
6     int n = 1;  
7     String w = "one";  
8     someFunction(n, w);  
9 }  
10  
11 void draw() {  
12 }  
13
```

The code uses color-coded syntax highlighting: `void`, `int`, and `String` are in green; `System.out.println` is in blue; and variable names like `num`, `word`, `n`, `w`, and function names like `someFunction`, `setup`, and `draw` are in black. The numbers 1 through 13 on the left are line numbers. Below the code editor, there is a light blue bar containing the word "Problem" on the left and "Tab" on the right.

# PrintTime Function Assignment

Complete the PrintTime Function Assignment by typing in code below plus your code individually. Submit to slack. Complete the Google Classroom Assignment

```
print_time ▾
1 void printEuropeanTime(int hour, int minute) {
2     // Your Java code
3 }
4
5 void printAmericanTime(int hour, int minute) {
6     // Your Java code
7 }
8
9 void setup() {
10    String dy = "Wednesday";
11    String mn = "September";
12    int dt = 20;
13    int yr = 1961;
14    int h = 7;
15    int m = 35;
16    // Insert Function call to printAmericanTime
17    // Insert Functioncall to printEuropeanTime
18 }
```

Problem	Tab	Line
The value of the parameter hour is not used	print_time	1
The value of the parameter minute is not used	print_time	1
The value of the parameter hour is not used	print_time	5
The value of the parameter minute is not used	print_time	5

# Unit 0-C

# Variables Types and Operators

# Unit 0-C

- Variables
- Types
- Declarations
- Initializations
- Comments
- % (modulus) and Integer division
- + and Strings

# Variables

- Think of a variable as a place to store a value that you will use later
- The value of a variable can *change* (think *vary*)
- A Java variable has size limits for its *type* (for example, integers are limited to values between -2,147,483,648 and 2,147,483,647)
- Every Java variable has a *type* and a *name*

# Variables: Parking space analogy

- Like a parking space, a variable can store a value (think *vehicle*) until you need it later



# Variables: Parking space analogy

- Parking spaces are often labeled so you can find your car later when you need it
- Lets say that **G210** is the *name* of this parking space



# Variables: Parking space analogy

- In addition to a *name*, parking spaces can have a *type* that sets size limits



# What is the error?

- We are trying to print an integer that is too large for Java's integer size

The screenshot shows the Processing IDE interface. The title bar reads "too\_large | Processing 4.0b2". The sketch window contains the following Java code:

```
1 void setup() {  
2     System.out.println(1234567890123456789);  
3 }  
4  
5 void draw() {  
6 }
```

The line `System.out.println(1234567890123456789);` is highlighted with a red underline, indicating a syntax error. Below the code editor, a red status bar displays the error message: "The literal 1234567890123456789 of type int is out of range". A detailed error log table follows:

Problem	Tab	Line
• The literal 1234567890123456789 of type int is out of range	too_large	2

At the bottom, there are tabs for "Console" and "Errors".

# Primitive data types

- In Java (unlike Python or JavaScript) variables have *types*
- Each type has size limits
- For this class, you need to know 5 basic (aka “primitive”) types:
  - **int**
  - **float**
  - **double**
  - **boolean**
  - **char**

# Primitive data types

- **int** holds a single integer value between -2,147,483,648 and 2,147,483,647
- **float** a decimal value with up to 7 digits
- **double** a decimal value with up to 15 digits
- A **boolean** can only hold values that evaluate to either **true** or **false**
- **char** holds a single letter, digit, space or punctuation mark and must be enclosed in *single quotes*, like this: 'G'

# float vs. double

- **float** is short for *floating point*, another name for *decimal point*
- **double** gets its name because of its size, it has about twice as many digits as a **float**

The screenshot shows the Processing 4.0b2 interface with a Java sketch titled "sketch\_220914e". The code compares the output of a `double` variable with a `float` variable:

```
1 double dNum = 5/9.0;
2 System.out.println(dNum);
3 System.out.println((float)dNum);
```

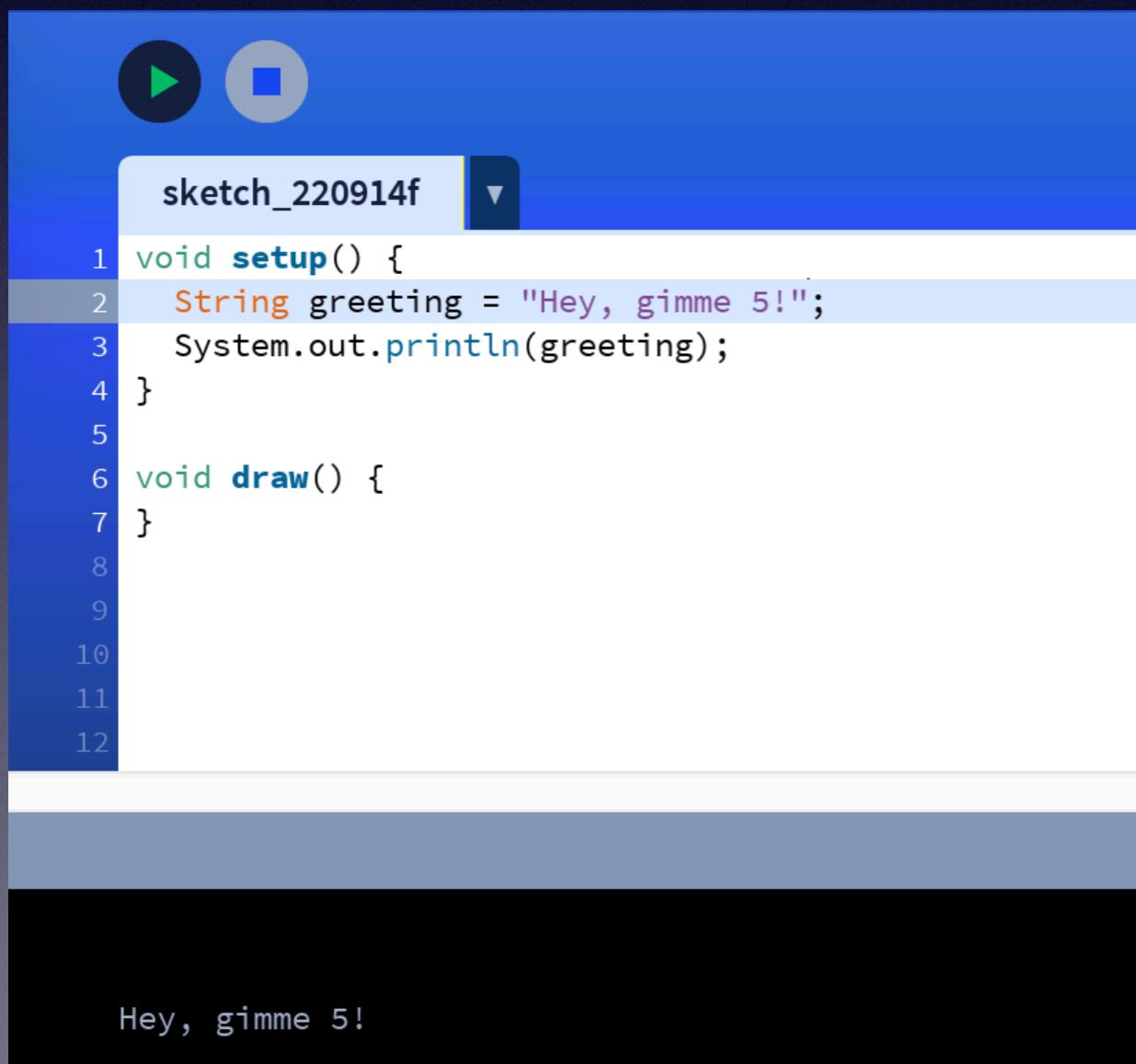
The console output shows two results:

```
0.5555555820465088
0.5555556
```

The "Console" tab is selected at the bottom.

# String variables

- A **String** variable can store text with any number of letters, digits, punctuation marks and spaces
- The beginning and end of the text is marked with double quotes ("")



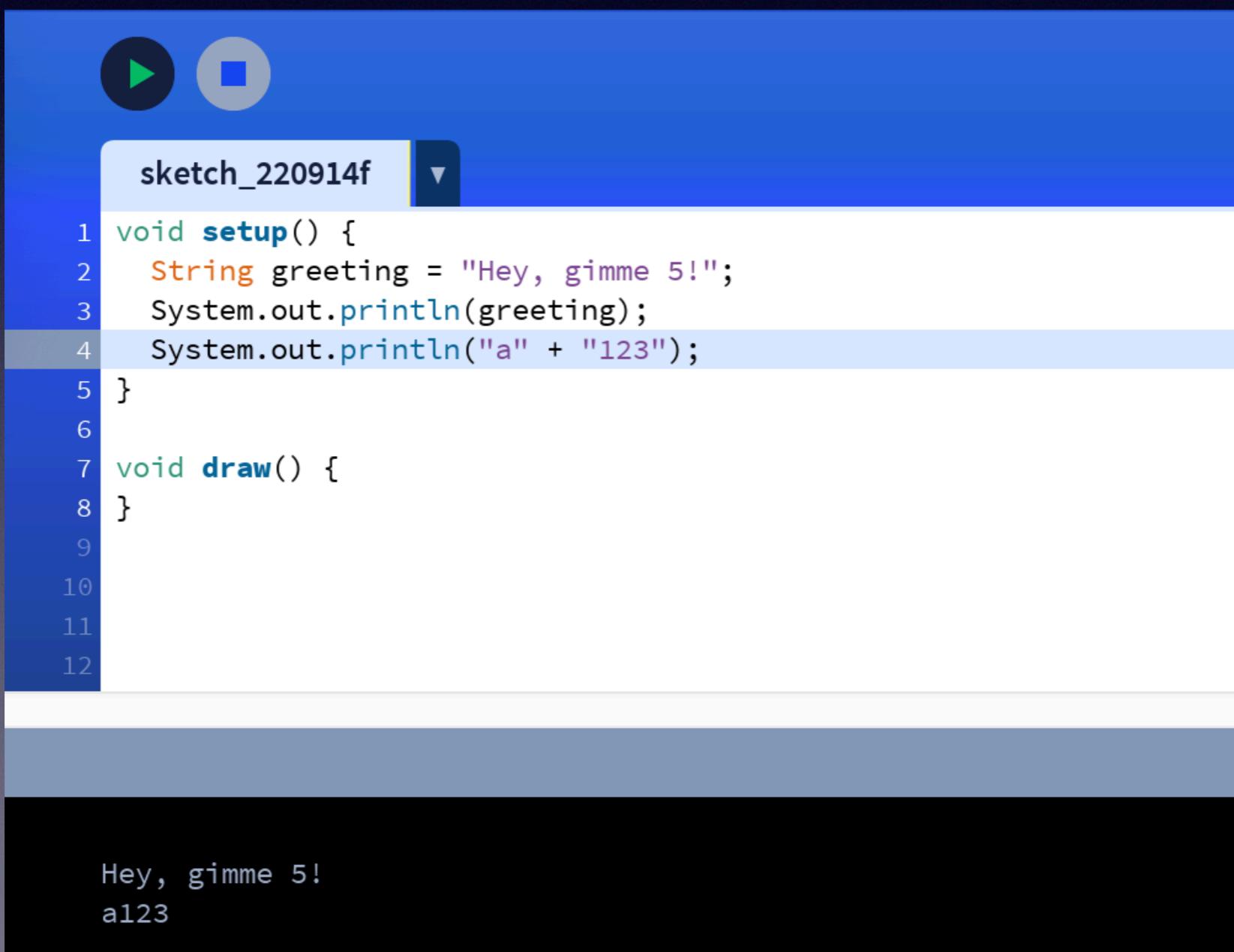
```
sketch_220914f

1 void setup() {
2   String greeting = "Hey, gimme 5!";
3   System.out.println(greeting);
4 }
5
6 void draw() {
7 }
8
9
10
11
12
```

Hey, gimme 5!

# + and Strings

- Using **+** with **Strings** isn't addition arithmetic, it's called *concatenation*
- That's a fancy word that means making bigger **Strings** out of little ones



The screenshot shows the Arduino IDE interface. At the top, there are two buttons: a green play button and a grey square button. Below them is a dropdown menu showing "sketch\_220914f". The main area contains the following code:

```
1 void setup() {  
2     String greeting = "Hey, gimme 5!";  
3     System.out.println(greeting);  
4     System.out.println("a" + "123");  
5 }  
6  
7 void draw() {  
8 }  
9  
10  
11  
12
```

Below the code, the output window displays the results of the println statements:

```
Hey, gimme 5!  
a123
```

# + and Strings

- Java executes from left to right so, `1 + 2` is `3`, and `3 + "Hello"` is `"3Hello"`
- `"Hello" + 1` is `"Hello1"`, and `"Hello1" + 2` is `"Hello12"`
- What's the output?

```
System.out.println(1 + 2 + "Hello");
```

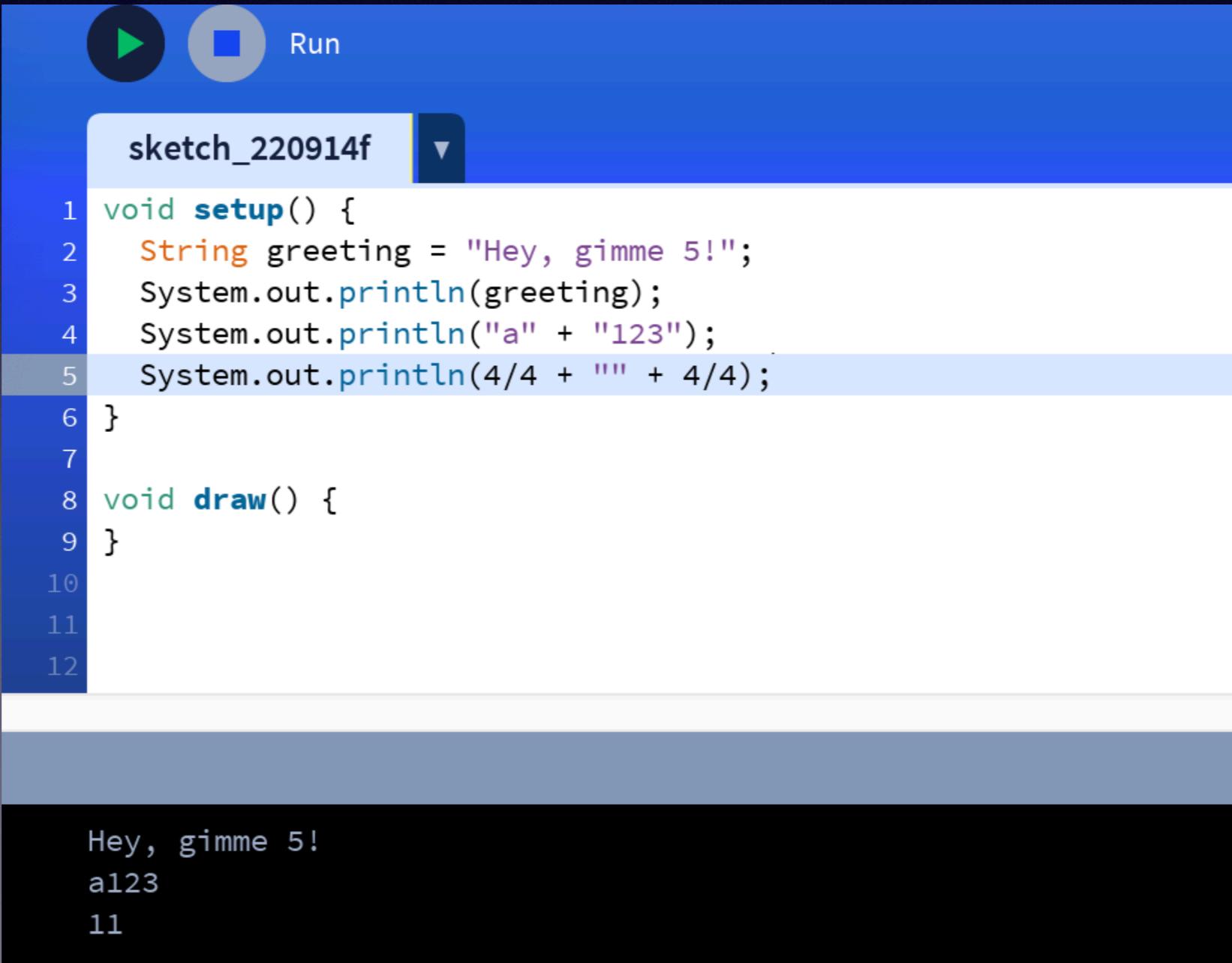
// the output is

```
System.out.println("Hello" + 1 + 2);
```

// the output is

# Unique way to make 11 using four 4s

- We could make 11 with four 4s by putting an empty **String** in the middle
- (**String** concatenation is not allowed in the rules of the four 4s challenge though)



```
sketch_220914f

1 void setup() {
2     String greeting = "Hey, gimme 5!";
3     System.out.println(greeting);
4     System.out.println("a" + "123");
5     System.out.println(4/4 + "" + 4/4);
6 }
7
8 void draw() {
9 }
10
11
12
```

Hey, gimme 5!  
a123  
11

# A Java variable declaration

- The Java statement that sets up a variable is called a *declaration*
- Here's an example declaration

```
int num;
```

- The first word of the declaration is the **type**
- The second word is the **name** that the programmer chooses

# Variable names and keywords

- You can pretty much choose any name you want for a variable with a few limitations:
  - Variable names cannot
    - start with a number
    - contain a space
    - have a special meaning in Java
  - Java has about 50 reserved words or *keywords* that you are not allowed to use as variable names such as **public**, **class**, **static**, **void**, **int** ...

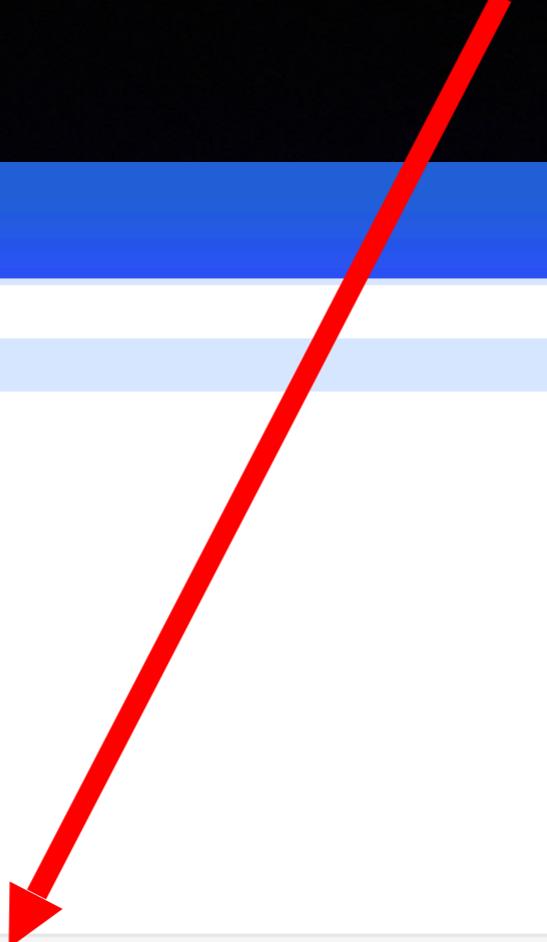
# camelCase

- Because a Java variable name can't have spaces, a style called **camelCase** is usually used for variable names with more than one word
- **camelCase** capitalizes the first letter of each word except the first word
- Examples: **firstName**, **numberOfDogs**
- Java variable names are case-sensitive, so **firstName** is not the same as **firstname** or **FirstName**.

# What's the error?

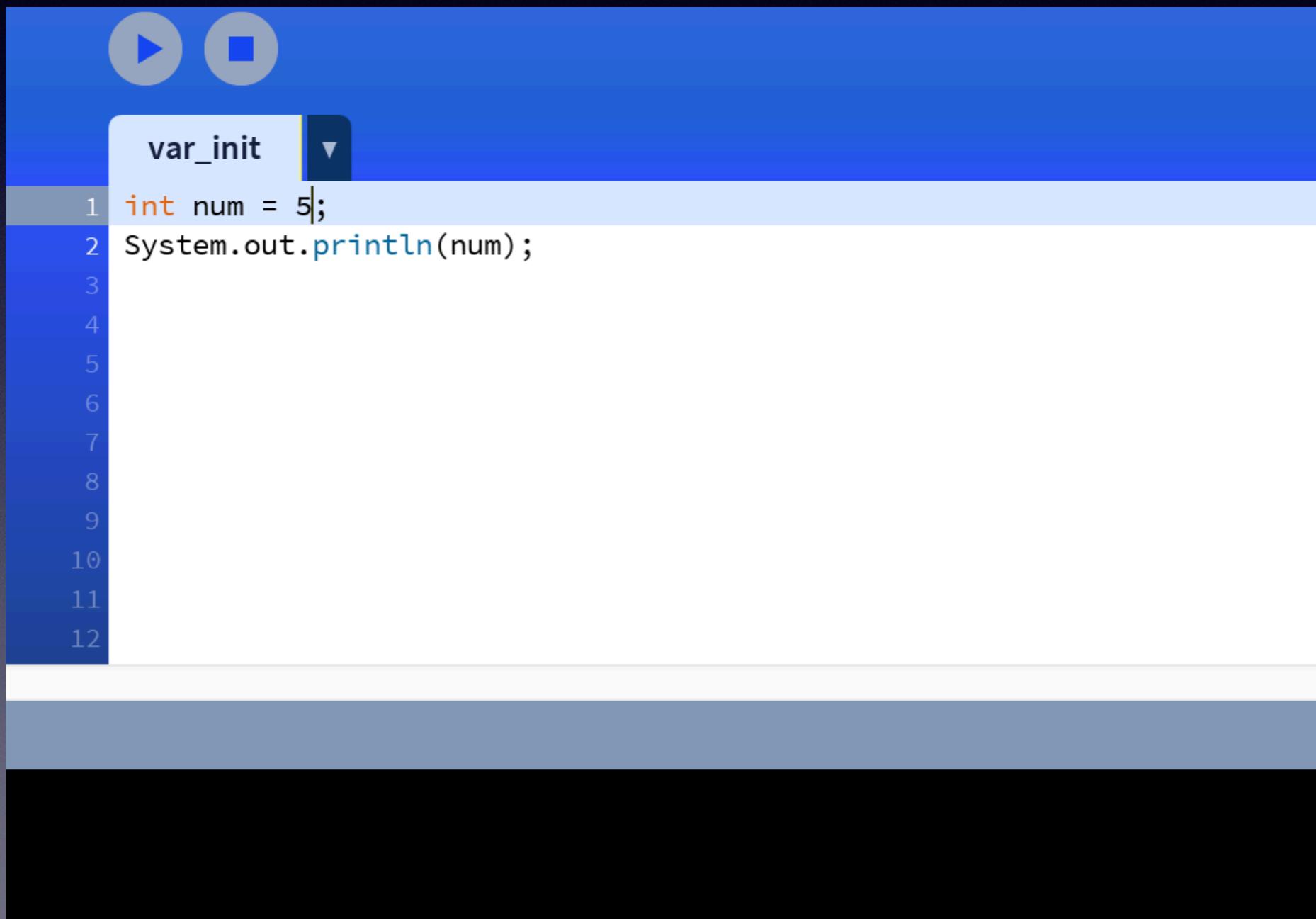
```
var_init ▾  
1 int num;  
2 System.out.println(num);  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12
```

The local variable "num" may not have been initialized



# Variable initialization

- After a Java variable is declared, it needs to be *initialized*



The image shows a screenshot of a Java code editor. At the top, there are two circular icons: a play button on the left and a square button on the right. Below them, the code is displayed:

```
var_init
1 int num = 5;
2 System.out.println(num);
3
4
5
6
7
8
9
10
11
12
```

The code consists of two lines: line 1 declares an integer variable 'num' and initializes it to 5; line 2 prints the value of 'num' to the console. Lines 3 through 12 are blank.

# Variable initialization

- The English word “initial” means *first*
- We initialize a variable by **assigning** (“setting it equal to”) its *first* value

The image shows a code editor window titled "var\_init". The code is as follows:

```
1 int num ;  
2  
3 void setup() {  
4     num = 17;  
5     System.out.println("num = " + num);  
6 }  
7  
8  
9  
10  
11  
12
```

A red arrow points from the text "its first value" in the slide's list to the assignment statement "num = 17" in the code editor.

At the bottom of the slide, there is a small preview of the code output: "num = 17".

# Declare *and* initialize

- You can **declare** and **initialize** a variable with one line of code:

```
int num = 3;
```

- Just remember that the one line of code is doing two different steps: the **declaration** and the **initialization**

# Comments

```
//Single Line
```

```
/*
```

```
Multi  
Line
```

```
*/
```

- Tells the computer to ignore some text

Used to:

- Write notes to yourself or other programmers

- Temporarily disable code

# Kahoot!

