# 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!

# Course Schedule

| LESSON UNITS | APPROXIMATE DURATION | TOPICS |
|---|---|---|
| **FALL 2022 SEMESTER** | **12-Weeks** | **Sep 2022 to Jan 2023** |
| **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 |
| **SPRING 2023 SEMESTER** | **15-Weeks** | **Jan to June 2023** |
| **Unit 6** | 2-Weeks | Arrays, Array Traversal, Data Structures, Project work |
| **Unit 7** | 3-Weeks | Array List, 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 APCSA 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 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

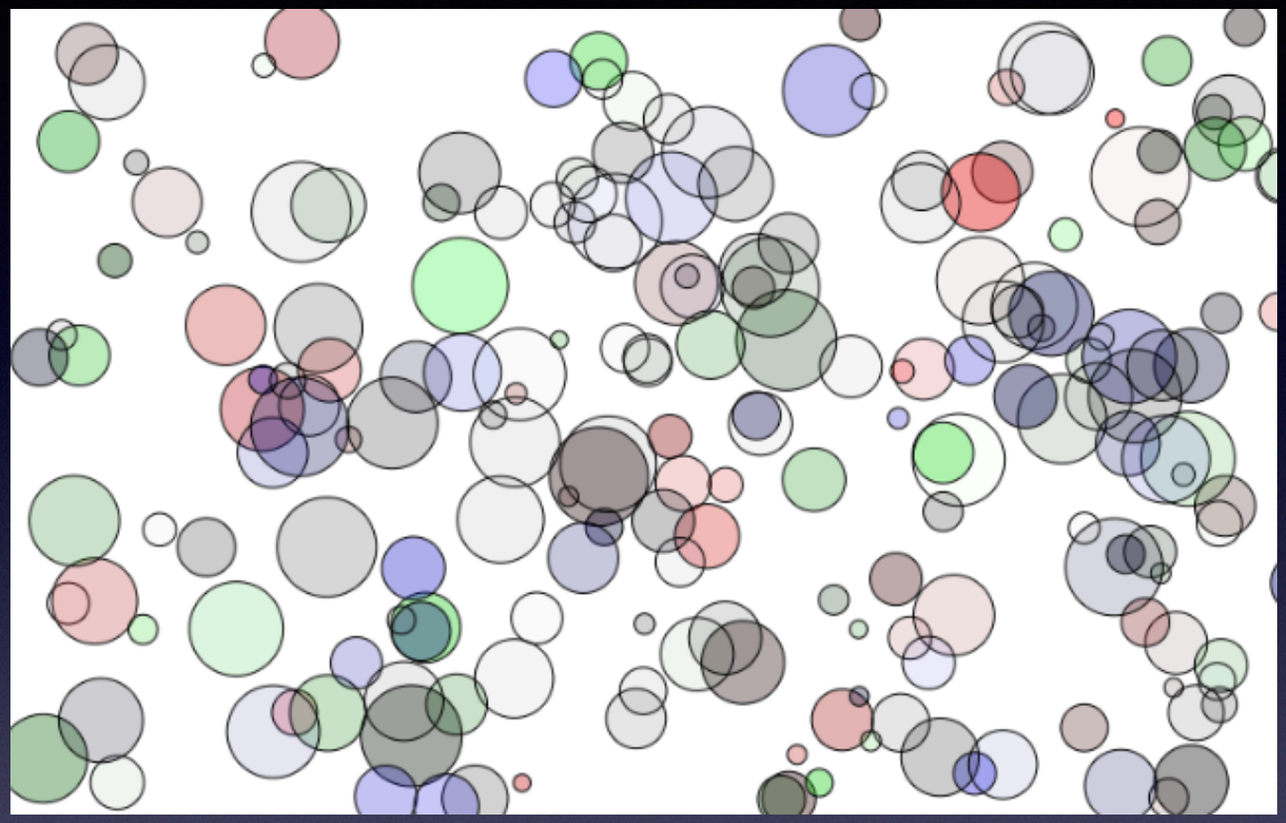
# System Requirements

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!



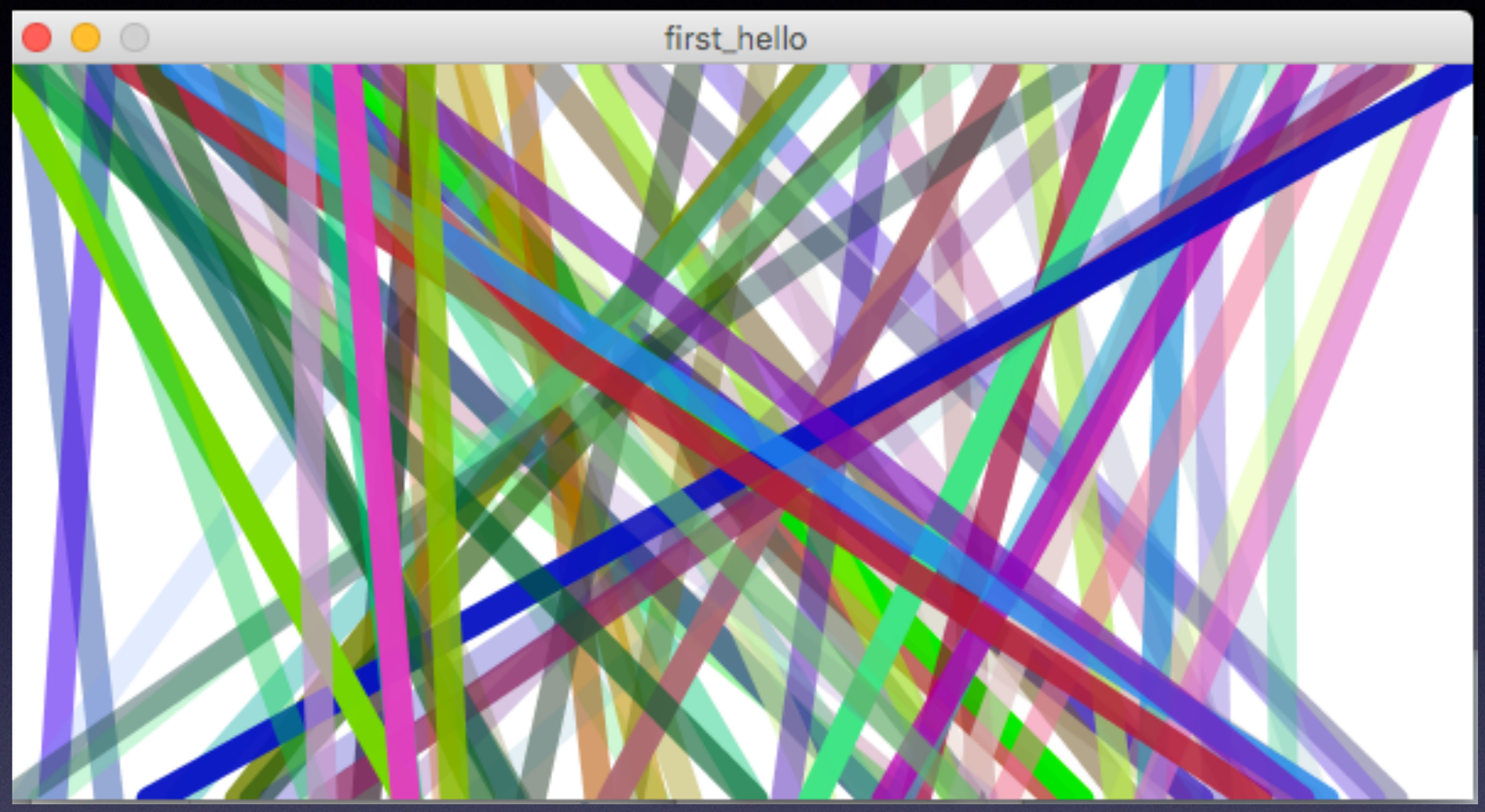
Bubbles - click to run the program

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

The common building blocks in programming languages are:

Variables

Loops

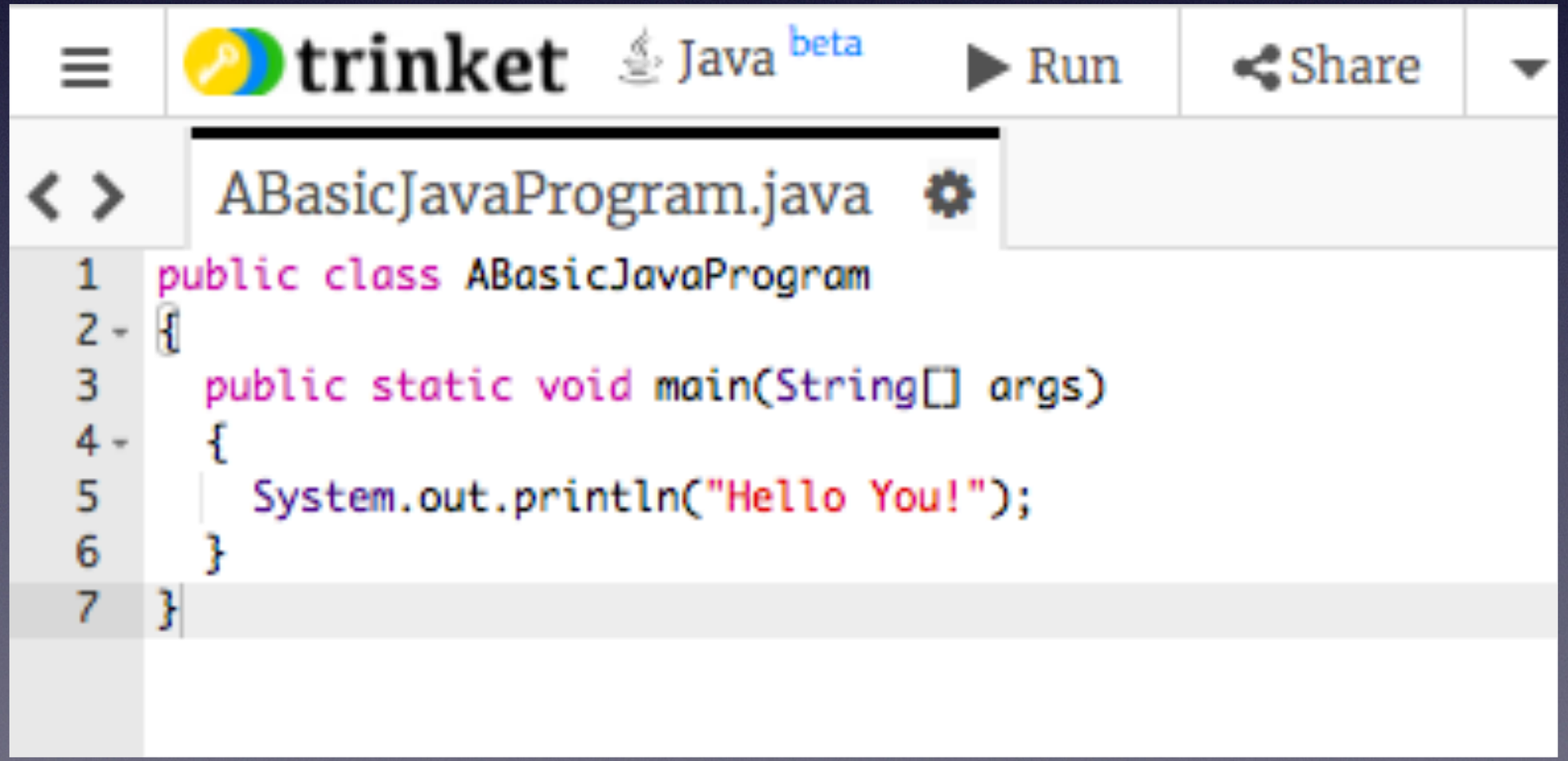
if statements

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 . . .

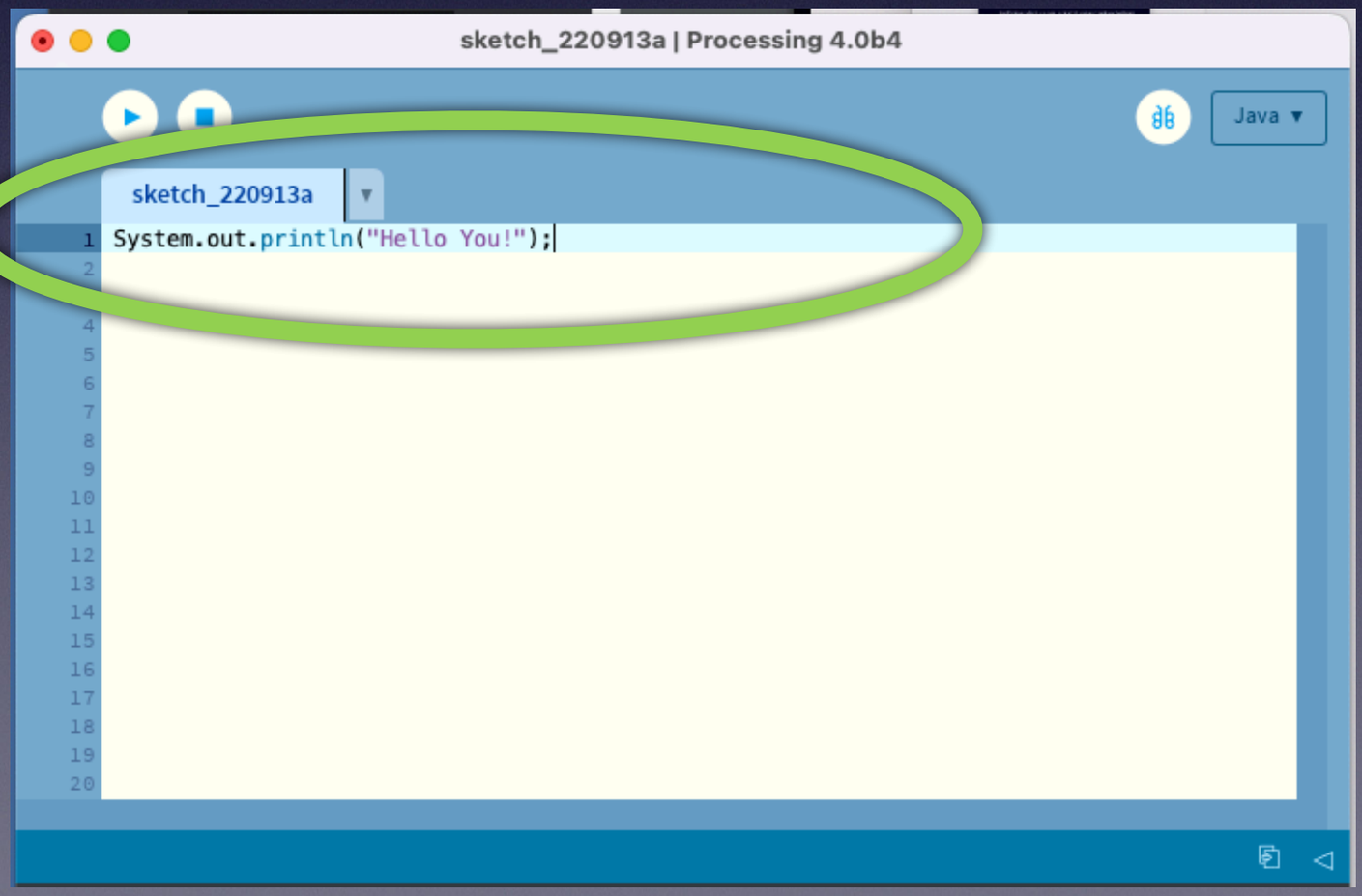


# A basic Java program

- 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!

# A basic Java program in Processing

- Much easier!



# 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

# 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 (")

# `print()` VS `println()`

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

TopBottom.java

```
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  }
9
```

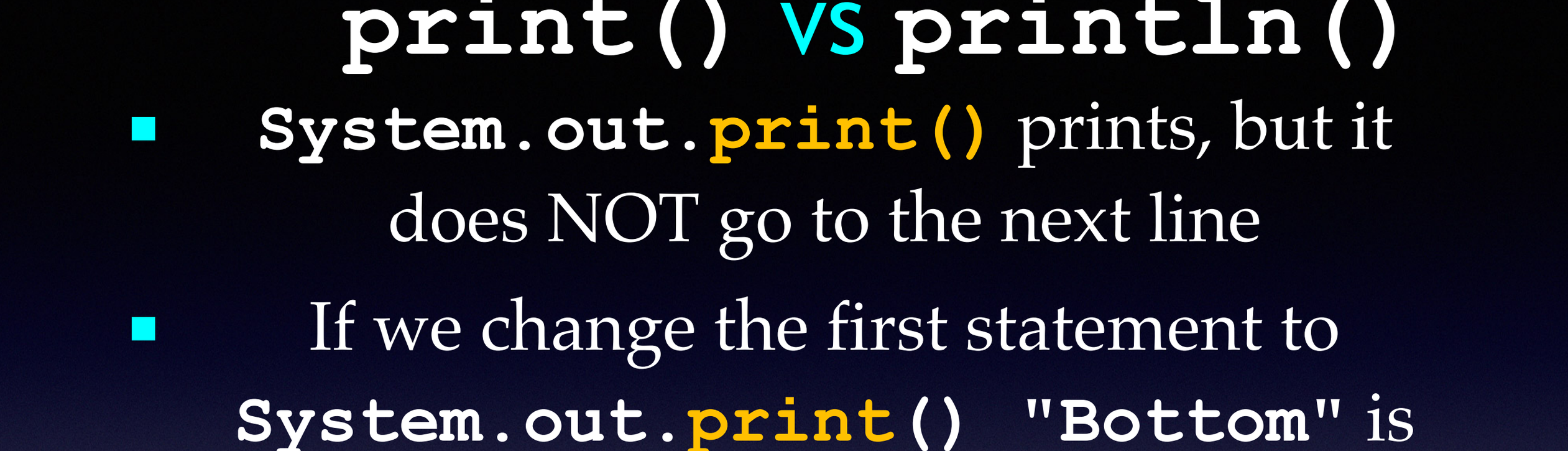
Result

Powered by trinket

```
Top
Bottom
```

# `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"`



# 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

Hello.java

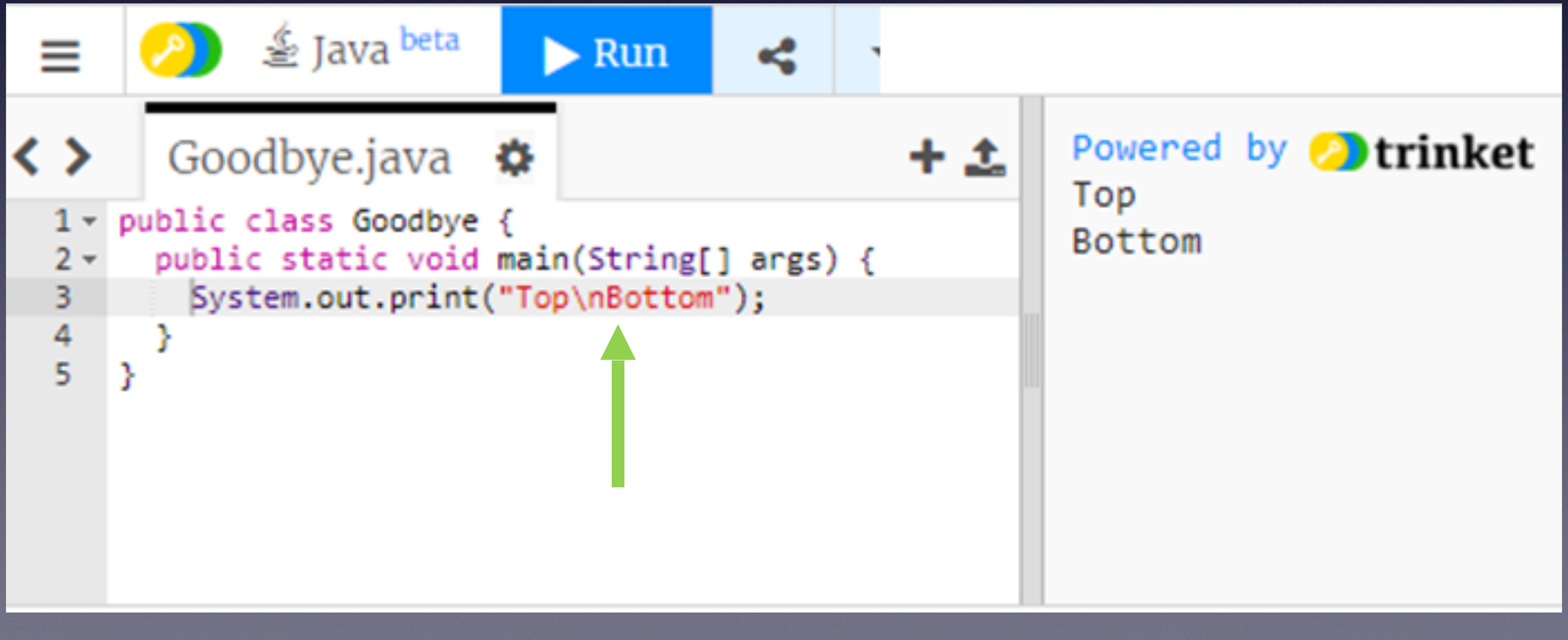
```
1  public class Hello {
2
3      public static void main(String[] args) {
4          // generate some simple output
5          System.out.println("Hello, World!");
6      }
7  }
```

# 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 ( / )*

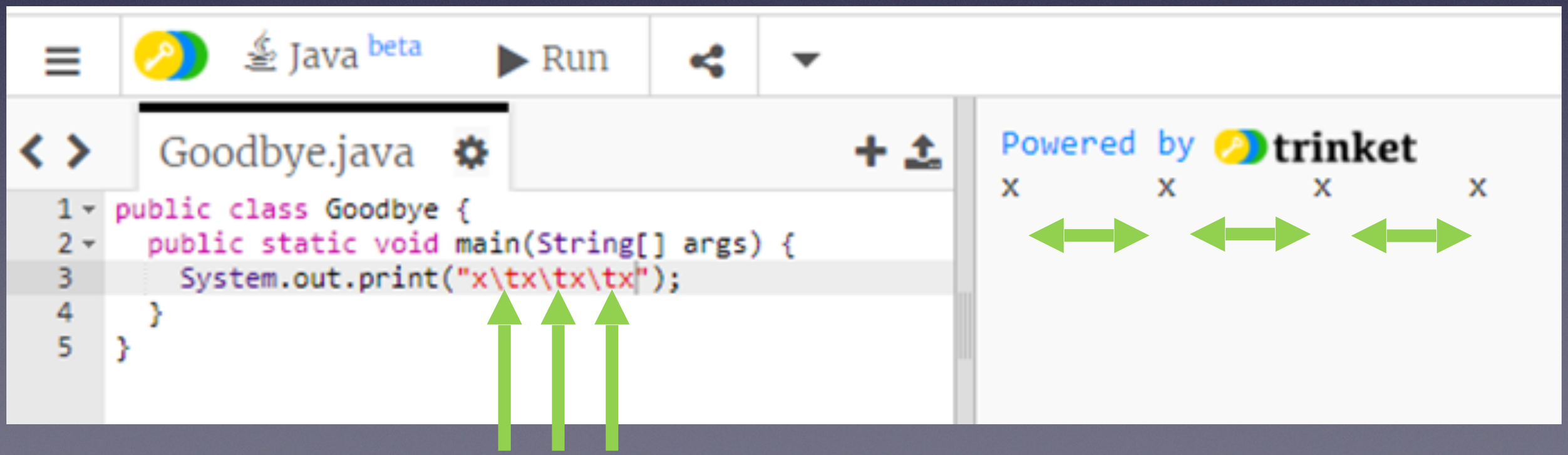
# Common Java Escape Sequences

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



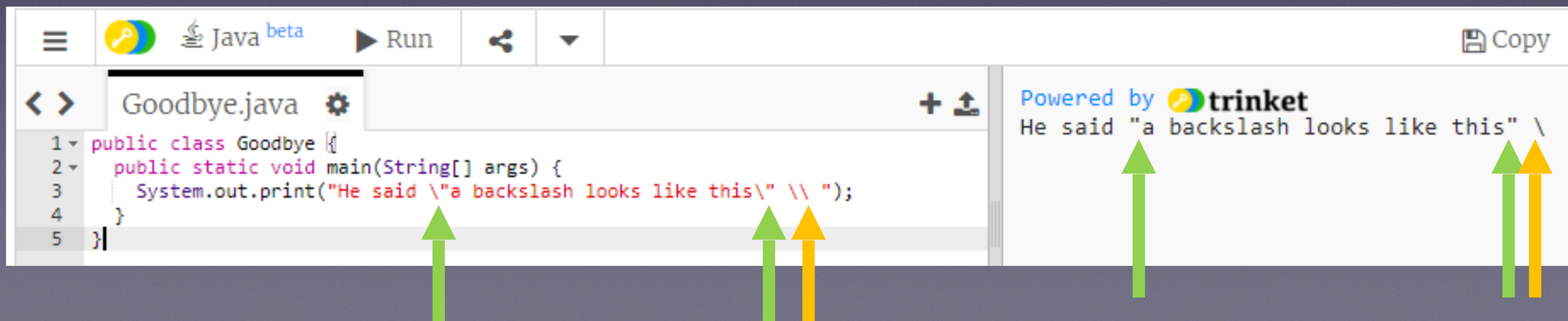
# Common Java Escape Sequences

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



# Common Java Escape Sequences

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

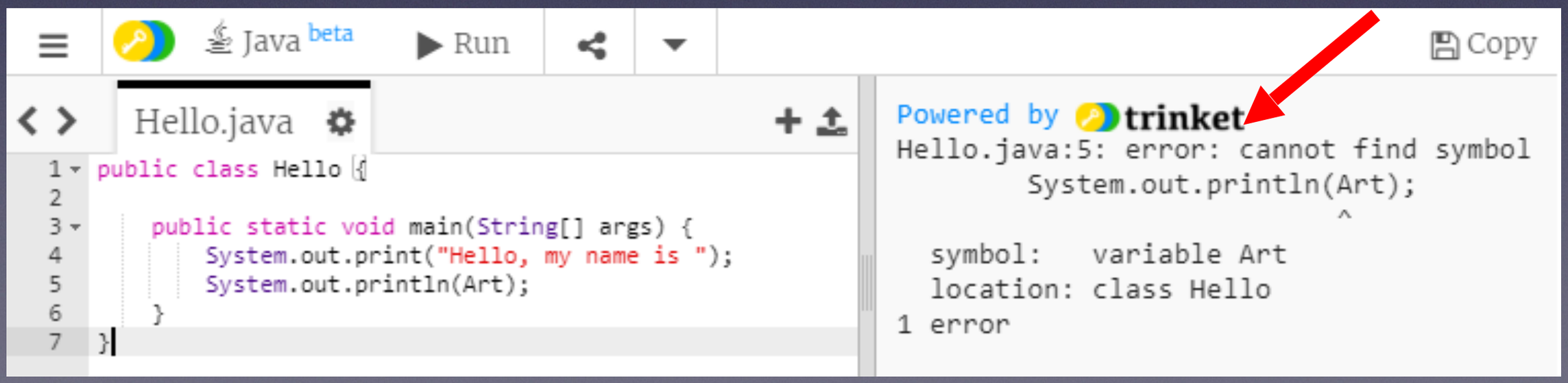


# Debugging

- 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

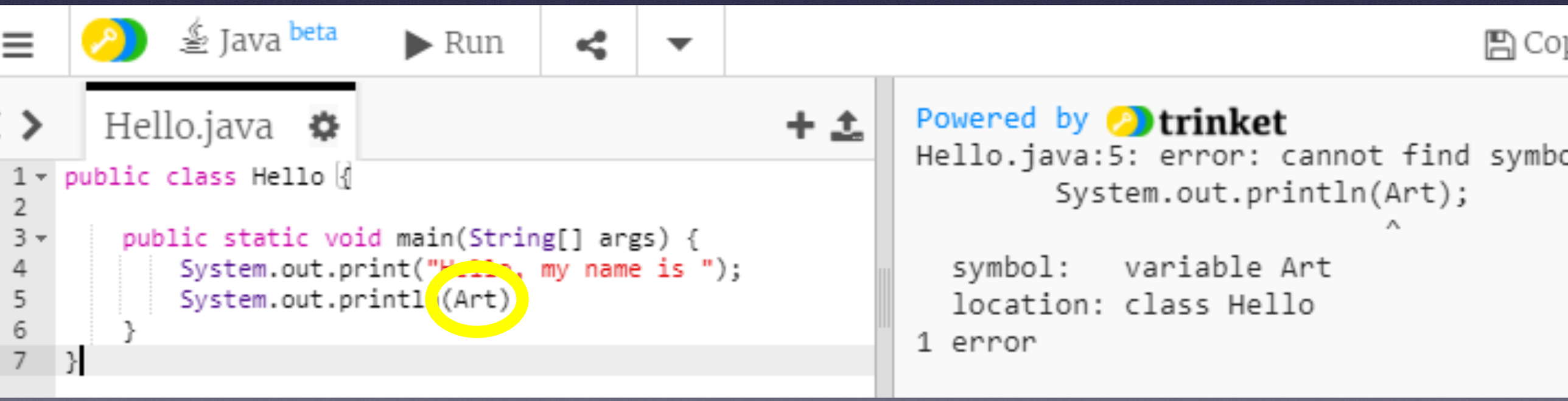
# Errors

- 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



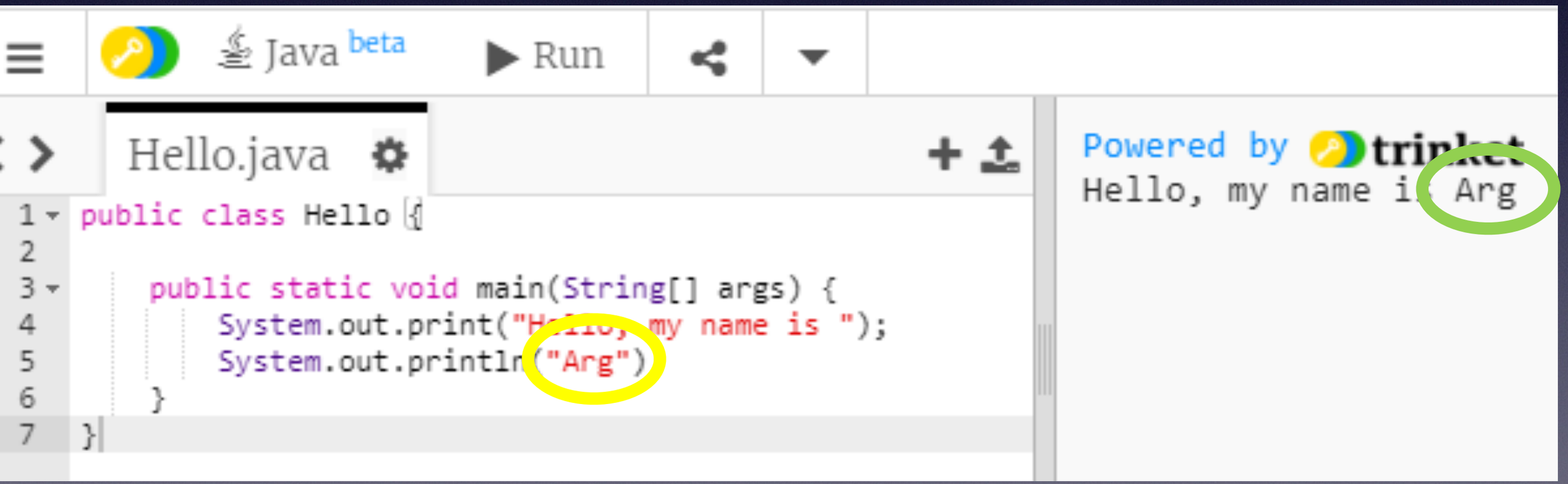
# Syntax error

- 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



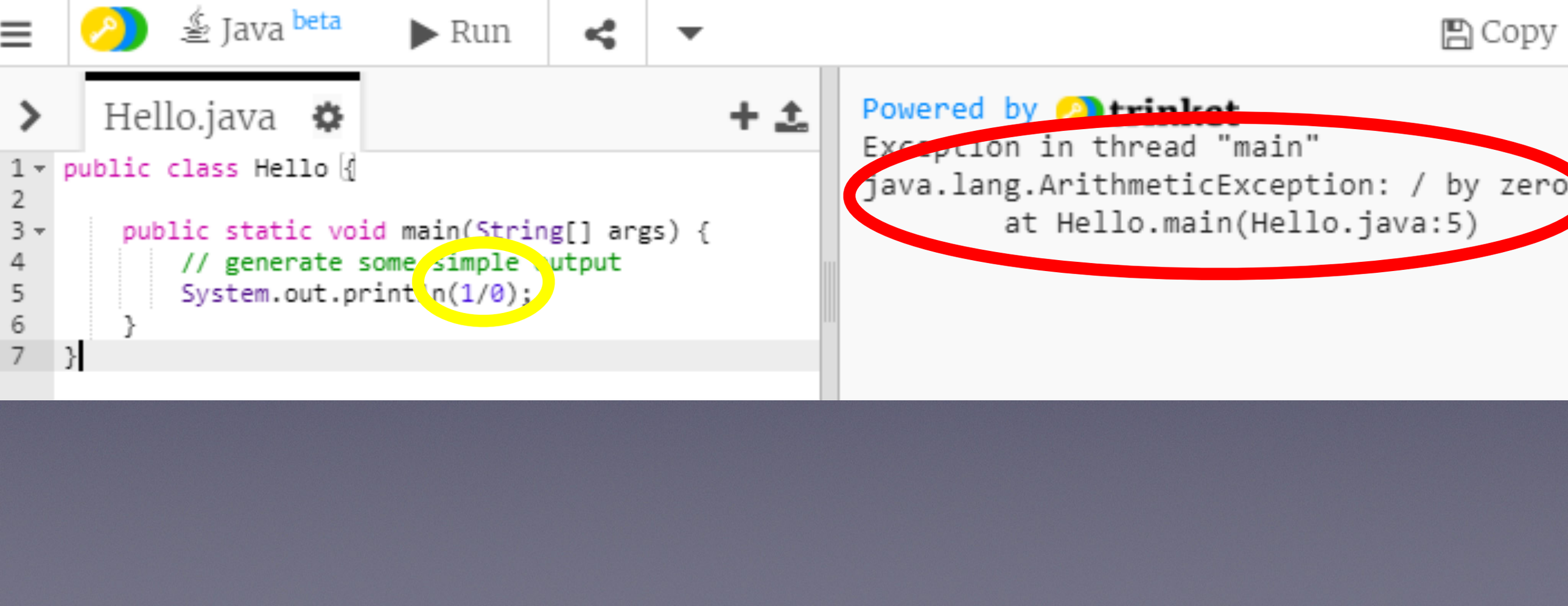
# Logic error

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



# (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



# Arithmetic in Java

+ - * /

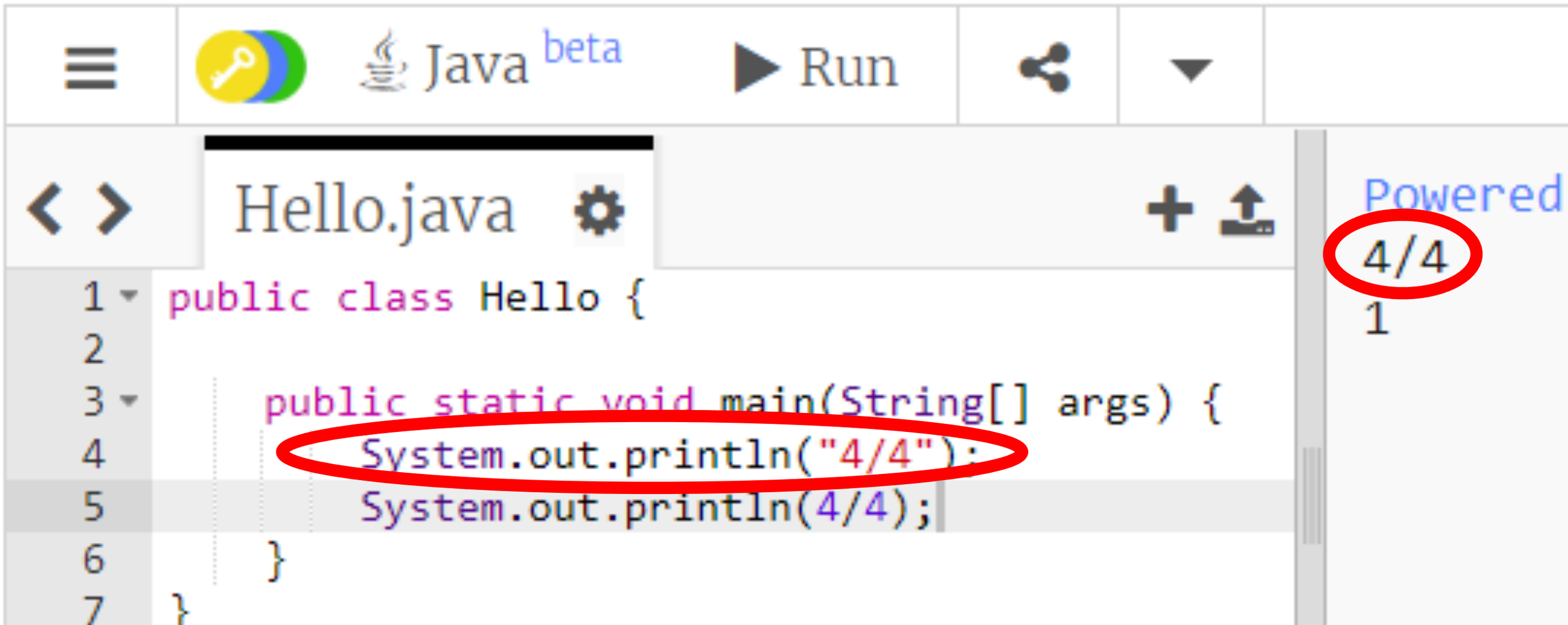
- Addition
- Subtraction
- Multiplication
- Division

# Literals vs. Expressions

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

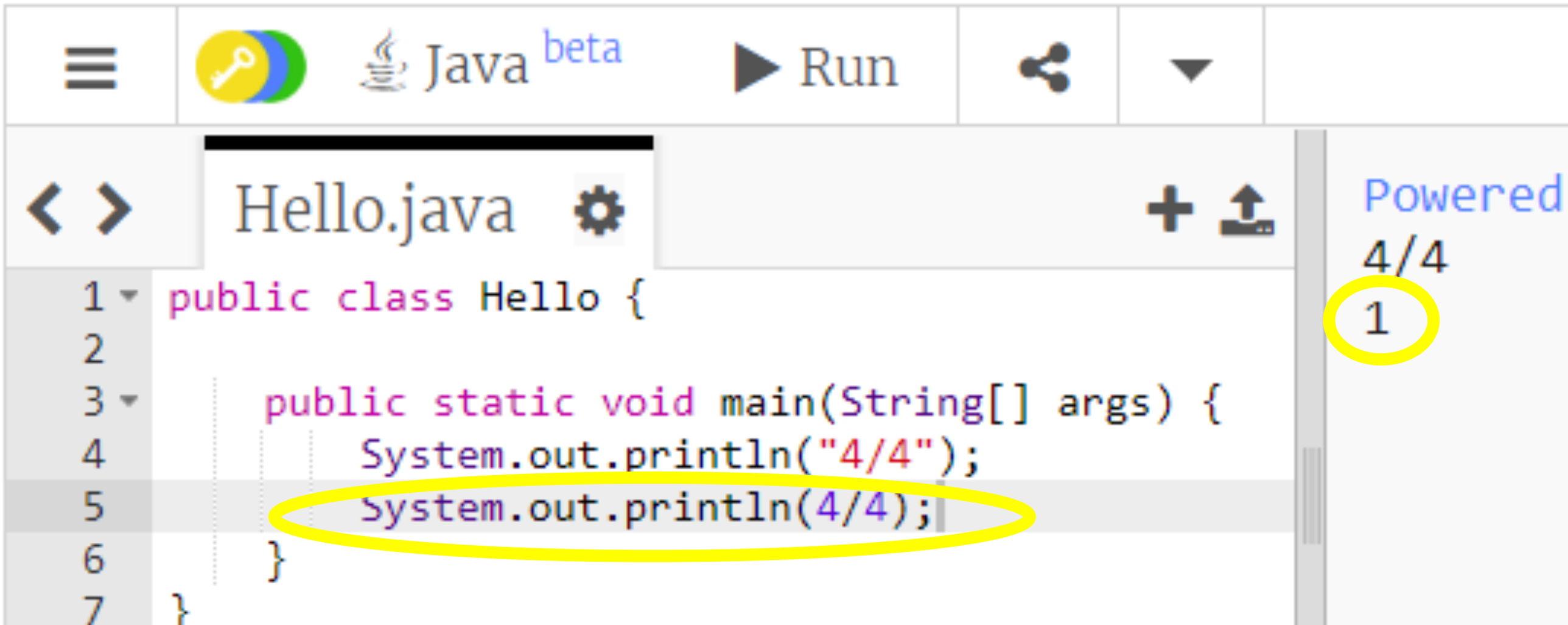
# Literals vs. Expressions

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



# Literals vs. Expressions

- Here's an expression `4 / 4`
- Java evaluates it to get an answer 1
- And then **`print`**s it

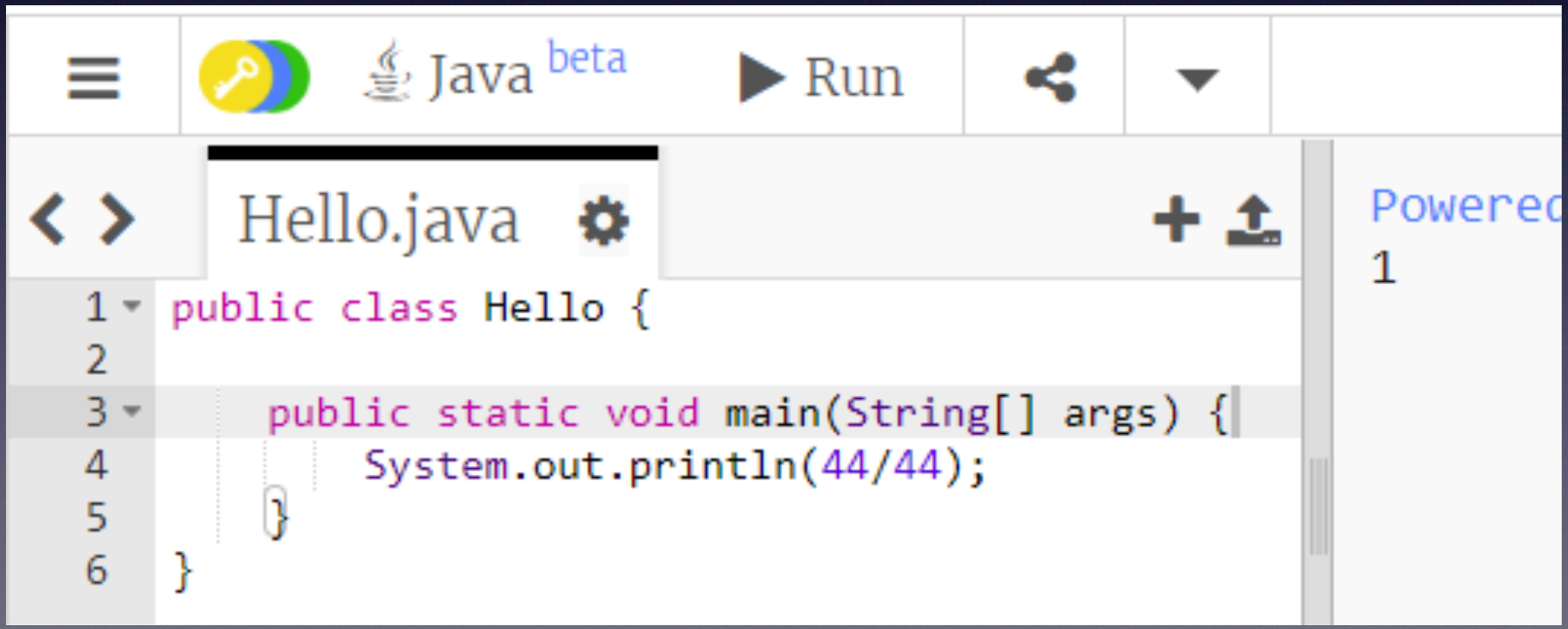


# Four 4s challenge

- 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.

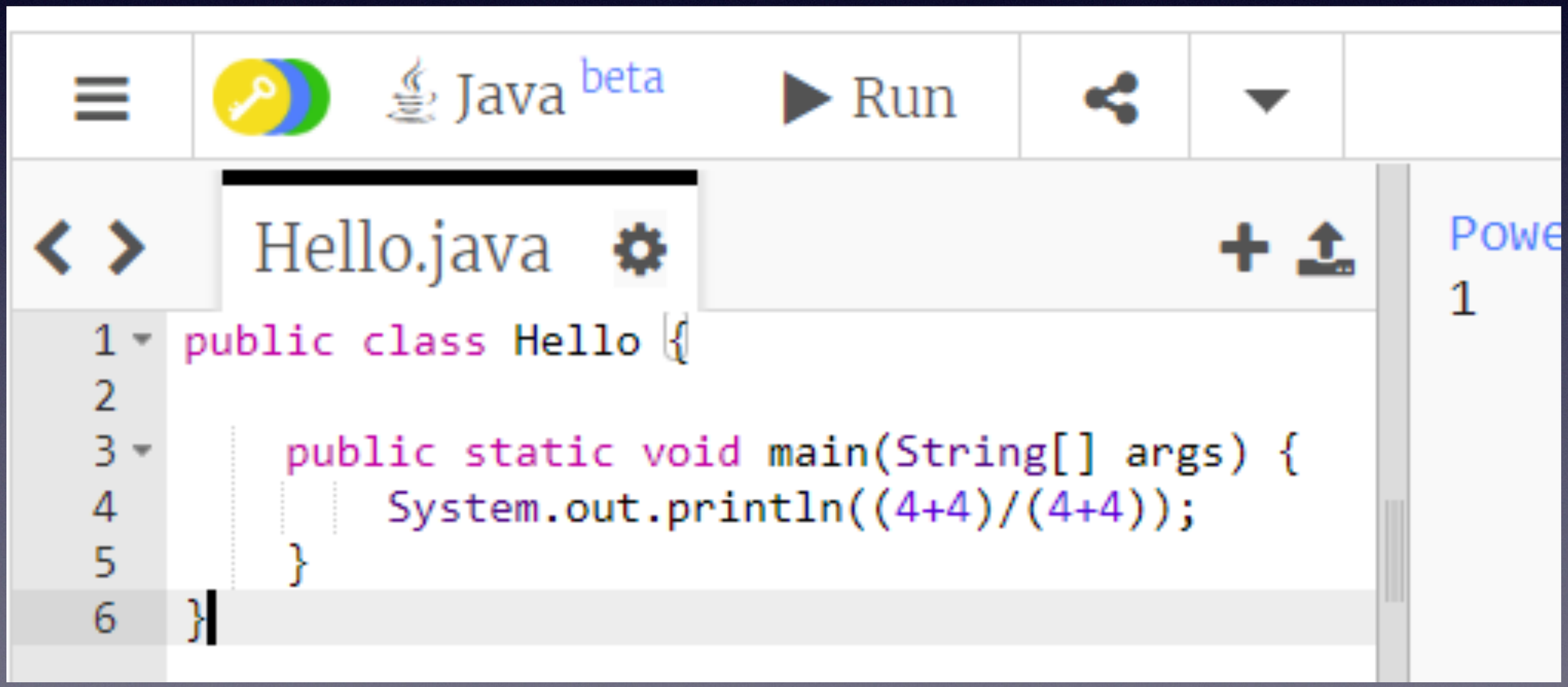
# Four 4s challenge

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



# Four 4s challenge

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



# Unit 1A

# 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();
```

argument

no argument

# Functions can be used for an *effect* 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 *value*s

# `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!`**

Java beta ▶ Run

NewLine.java

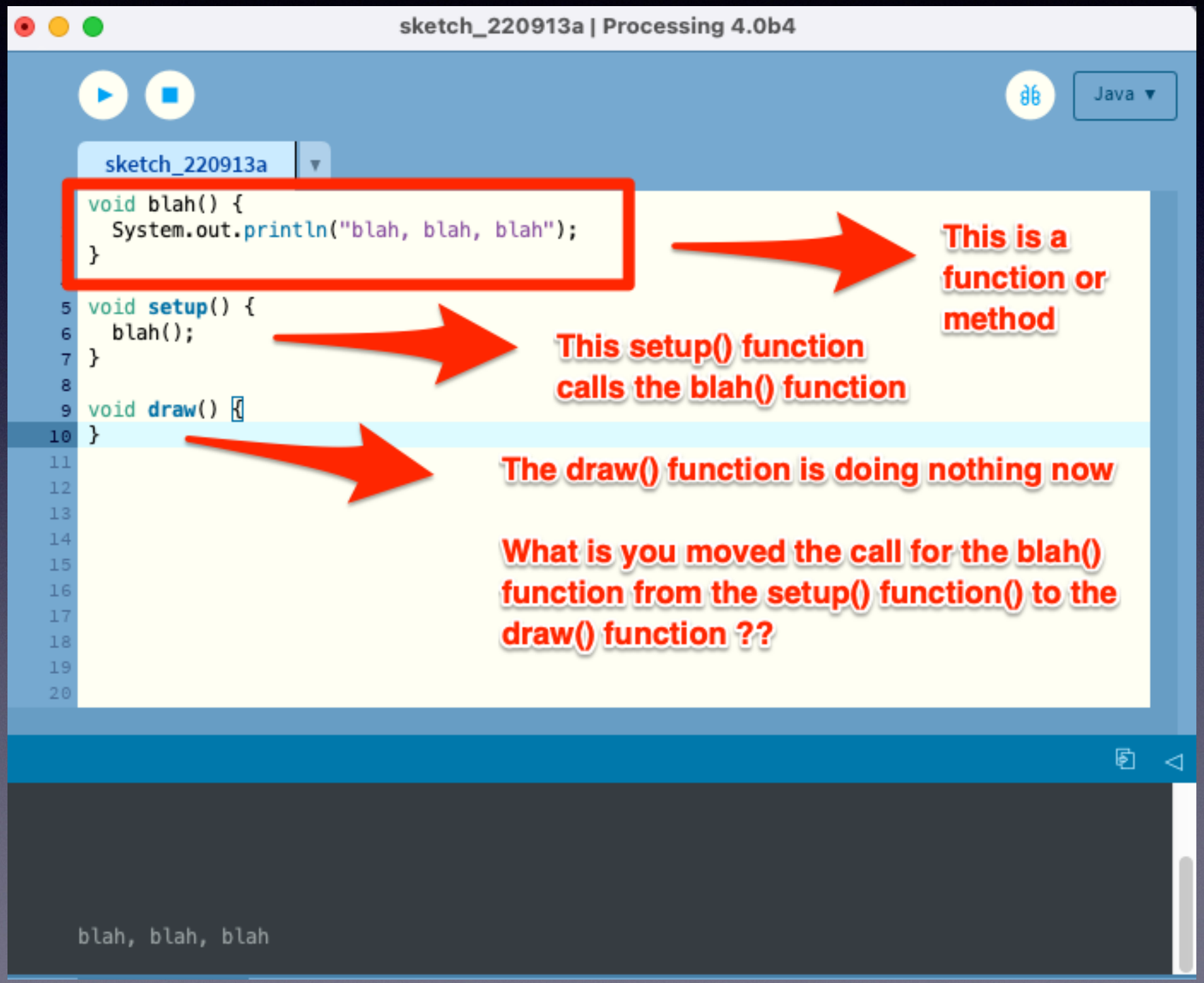
```
1  public class NewLine {
2
3      public static void blah() {
4          System.out.println("blah, blah, blah");
5      }
6
7      public static void main(String[] args) {
8          blah();
9      }
10 }
```

Powered by trinket

```
blah, blah, blah
```

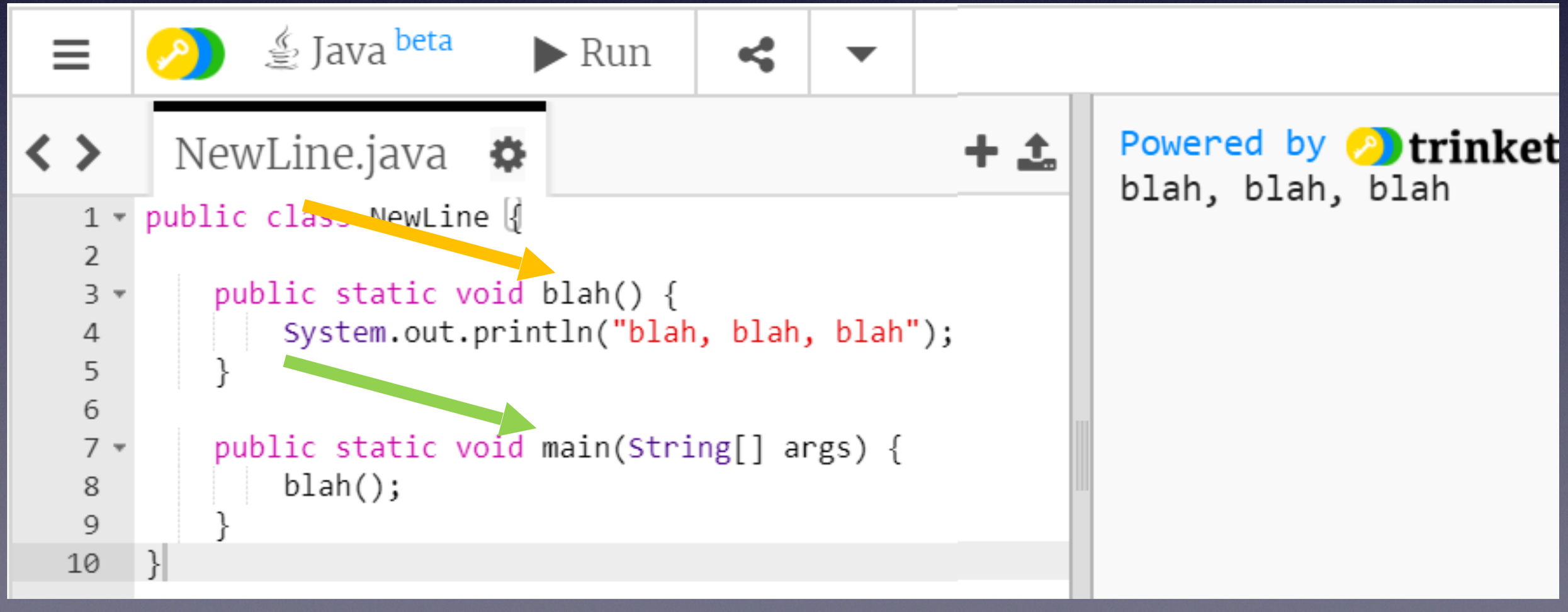
# `void` functions (aka methods)

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



# `void` functions (aka methods)

- Functions are like paragraphs of computer code
- Every Java program has a `main` function
- There can be many other functions



# `void` functions (aka methods)

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

Java beta ▶ Run

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)

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

Java beta ▶ Run

NewLine.java

```
1  public class NewLine {
2
3      public static void main(String[] args) {
4          blah();
5      }
6
7      public static void blah() {
8          System.out.println("blah, blah, blah");
9      }
10
11 }
```

Powered by trinket

```
blah, blah, blah
```

# `void` functions (aka methods)

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

Java beta ▶ Run

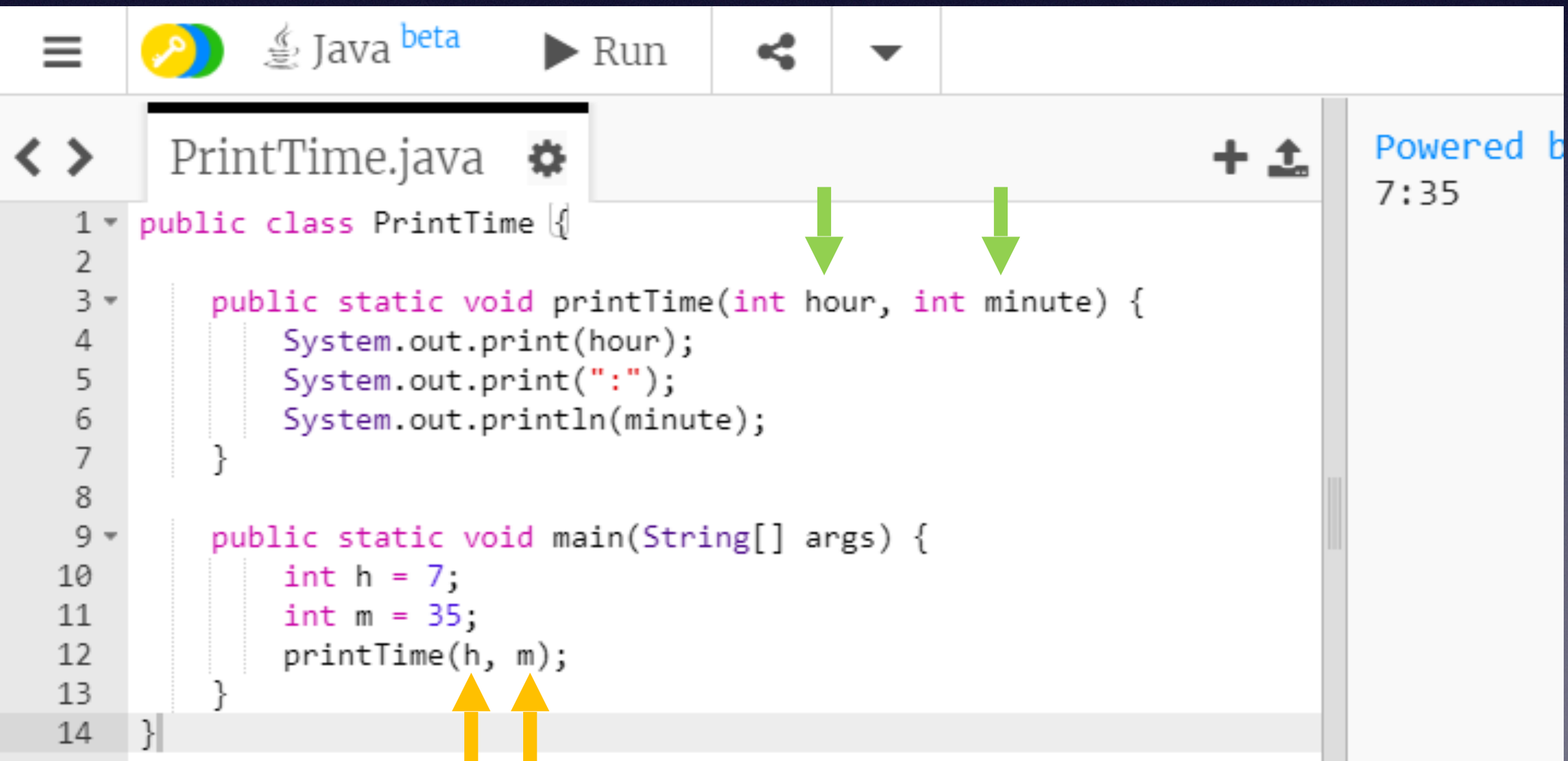
NewLine.java

```
1  public class NewLine {
2
3      public static void main(String[] args) {
4          blah();
5          boring();
6          blah();
7      }
8
9      public static void blah() {
10         System.out.println("blah, blah, blah");
11     }
12     public static void boring() {
13         System.out.println("boring, boring, boring");
14     }
15 }
```

Powered by trinket

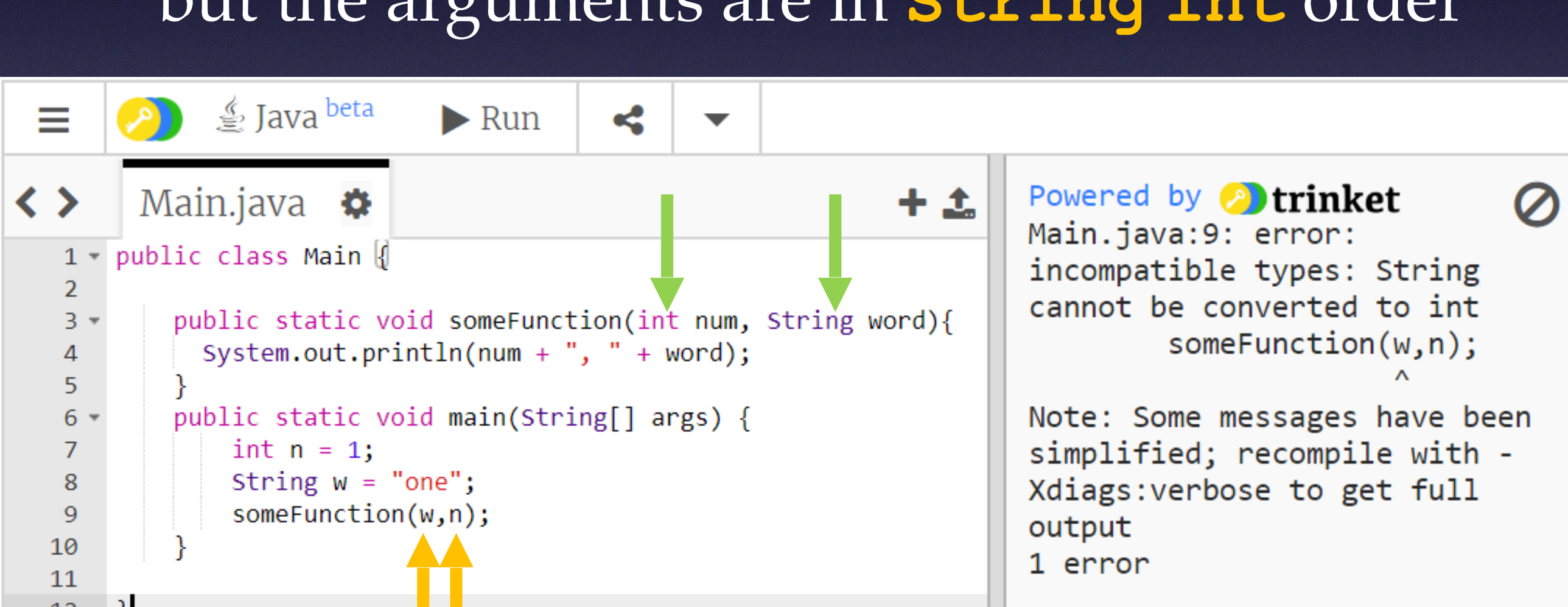
```
blah, blah, blah
boring, boring, boring
blah, blah, blah
```

- 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`**



# 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



# 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**

Main.java

```
1  public class Main {
2
3      public static void someFunction(int num, String word){
4        System.out.println(n + ", " + w);
5      }
6      public static void main(String[] args) {
7          int n = 1;
8          String w = "one";
9          someFunction(n,w);
10     }
11
12 }
```

Powered by trinket

```
Main.java:4: error: cannot
find symbol
        System.out.println(n +
", " + w);
                           ^
  symbol:   variable n
  location: class Main
Main.java:4: error: cannot
find symbol
        System.out.println(n +
", " + w);
```

# Now the program is correct

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

Main.java

```
1  public class Main {
2
3      public static void someFunction(int num, String word){
4        System.out.println(num + ", " + word);
5      }
6      public static void main(String[] args) {
7          int n = 1;
8          String w = "one";
9          someFunction(n,w);
10     }
11
12 }
```

Powered by trinket

```
1, one
```

# PrintTime Function Assignment

Complete the PrintTime Function Assignment from the Functions and Parameters section

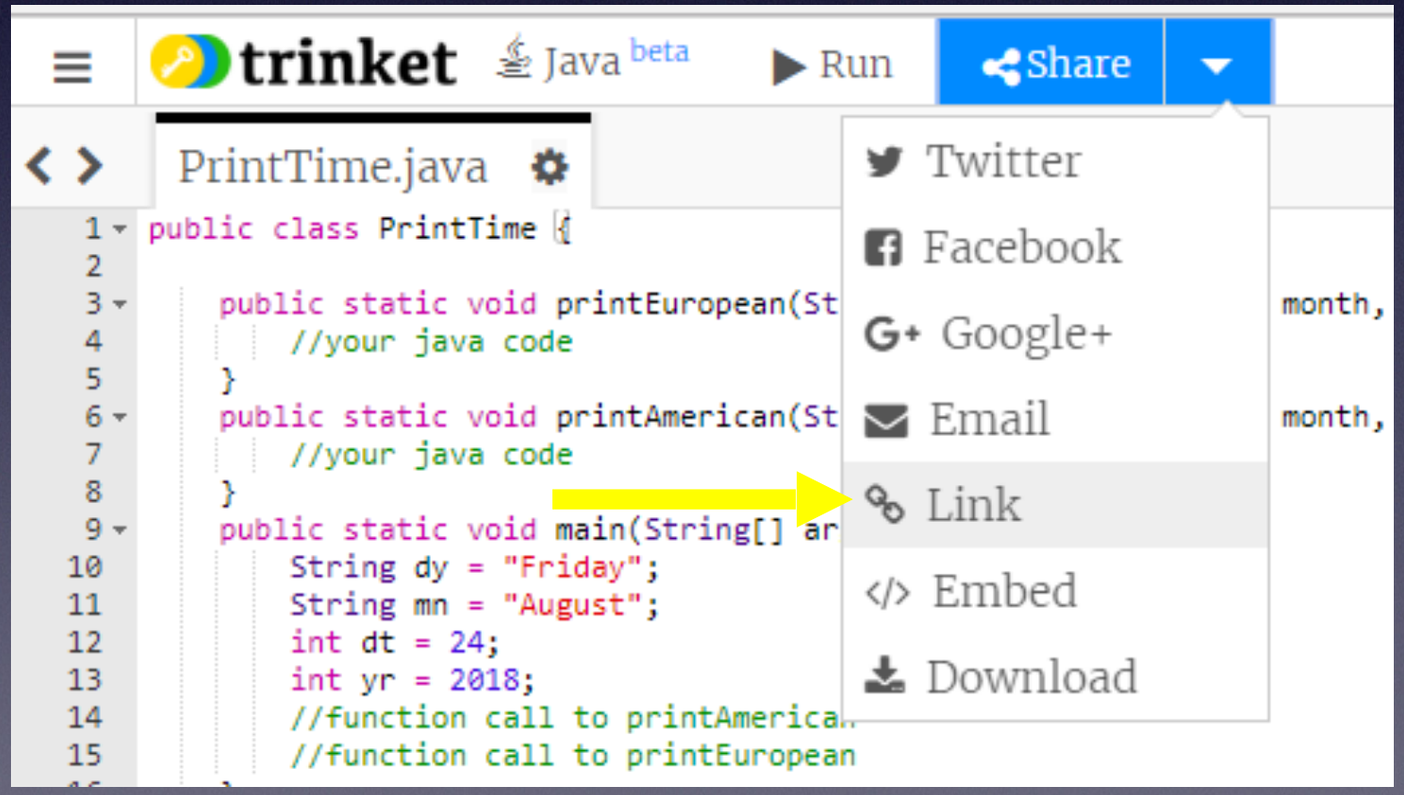
trinket Java beta ▶ Run Share

PrintTime.java

```
1  public class PrintTime {
2
3      public static void printEuropean(String day,int date,String month, int year) {
4          //your java code
5      }
6      public static void printAmerican(String day,int date,String month, int year) {
7          //your java code
8      }
9      public static void main(String[] args) {
10         String dy = "Friday";
11         String mn = "August";
12         int dt = 24;
13         int yr = 2018;
14         //function call to printAmerican
15         //function call to printEuropean
16     }
17 }
```

```
Powered by trinket
Friday, August 24, 2018
Friday, 24 August, 2018
```

Submit to google classroom by choosing *Share* | *Link* and submitting the link to google classroom



# Unit 1B
# Variables Types and Operators

# Unit 1B

- Variables
- Types
- Declarations
- Initializations
- Comments
- `%` (modulus) and Integer division
- `+` and **`String`**s

# 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

Java beta ▶ Run

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          // generate some simple output
5          System.out.println(123123123123123123232313);
6      }
7  }
```

Copy

Powered by trinket

```
Hello.java:5: error: integer number too large:
123123123123123123232313

System.out.println(123123123123123123232313);
                   ^

1 error
```

# 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`

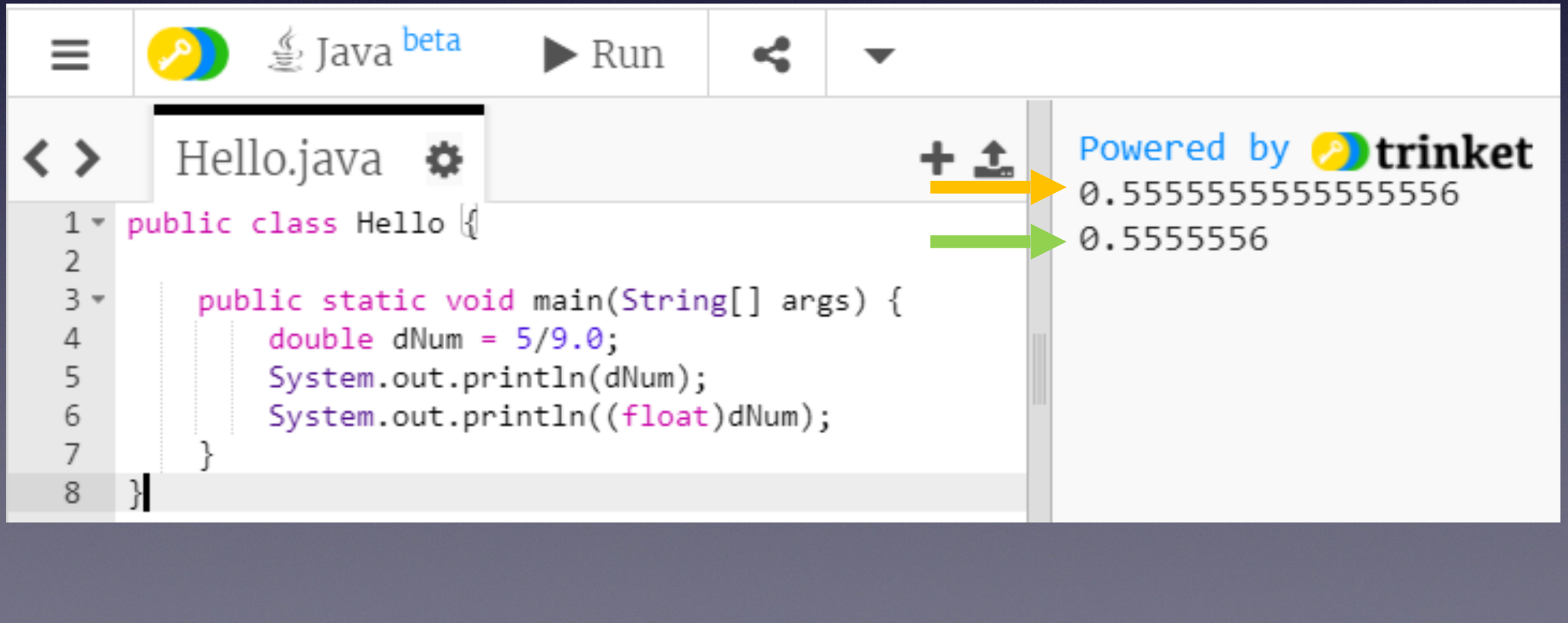
# Primitive data types

- **`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'`**

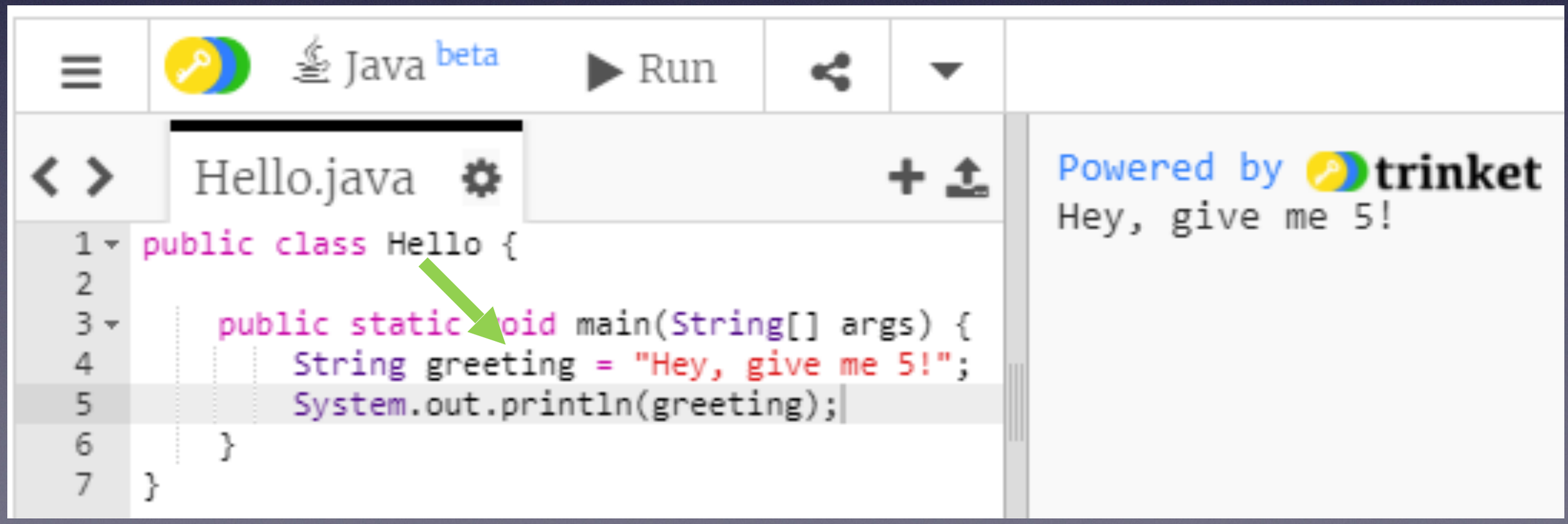
# `float` vs. `double`

- `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`



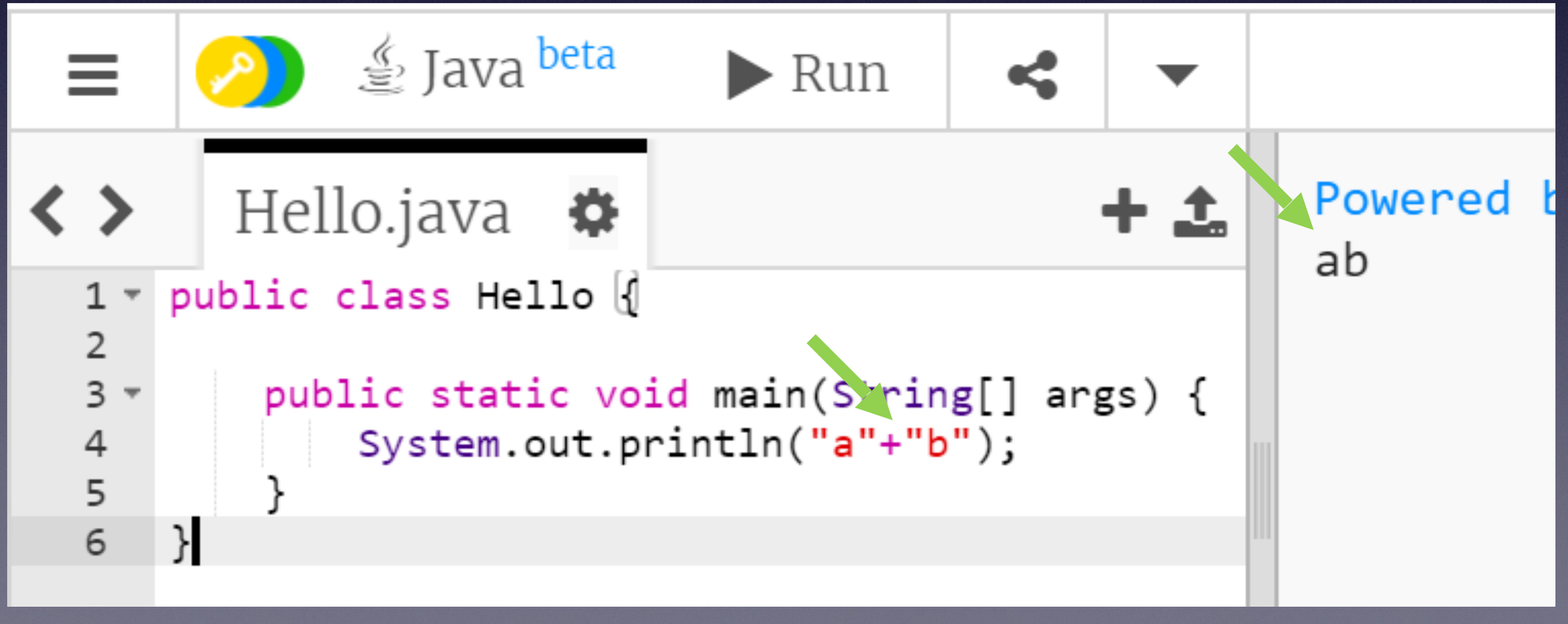
# `String` variables

- 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 (")



# + 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



# + and Strings

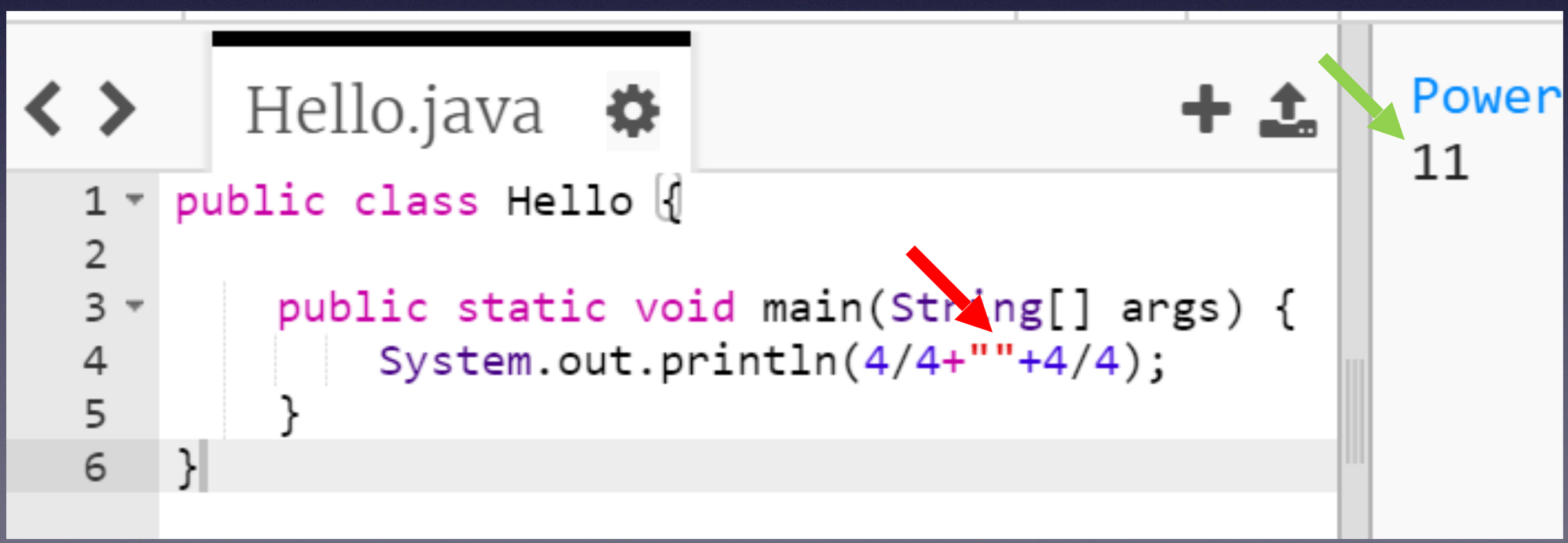
- Java executes from left to right so, `1 + 2` is `3`, and `3 + "Hello"` is `"3Hello"`
- `"Hello" + 1` is `"Hello1"`, and `"Hello1" + 2` is `"Hello12"`

```
System.out.println(1 + 2 + "Hello");
// the output is 3Hello

System.out.println("Hello" + 1 + 2);
// the output is Hello12
```

# 11

- 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)*



# 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

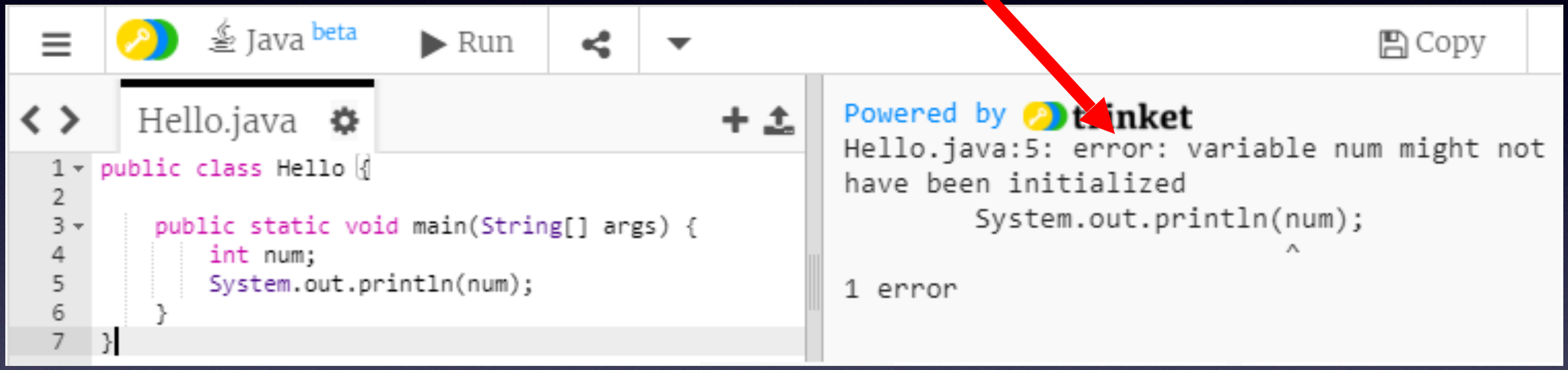
# Variable names and keywords

- 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

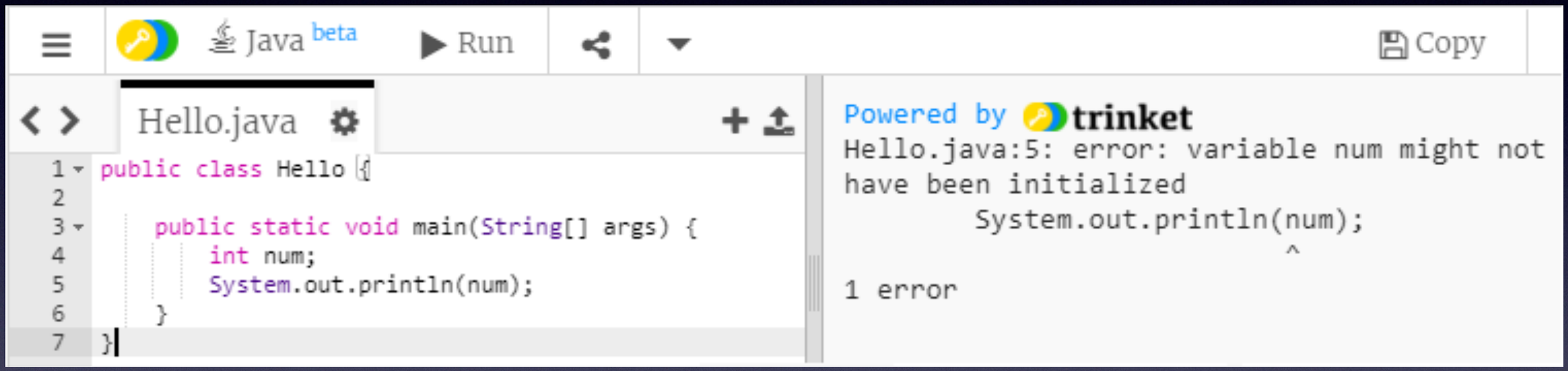
- Because a Java variable name can't have spaces, a style called **camelCase** is usually used for variable names with more than one word
- **camelCase** capitalizes the first letter of each word except the first word
- Examples: **`firstName`**, **`numberOfDogs`**
- Java variable names are case-sensitive,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

# What's the error?



# Variable initialization

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



# Variable initialization

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

# 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!