Java Intro

Basics Lesson
Chapter 1 - Variables Loops Functions Branching

Java Basics

The common building blocks in programming languages are:

Variables

Loops

if statements

Functions (aka “methods”)

It’s expected that you have done programming with these already in some language

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

A basic Java “Hello World” program

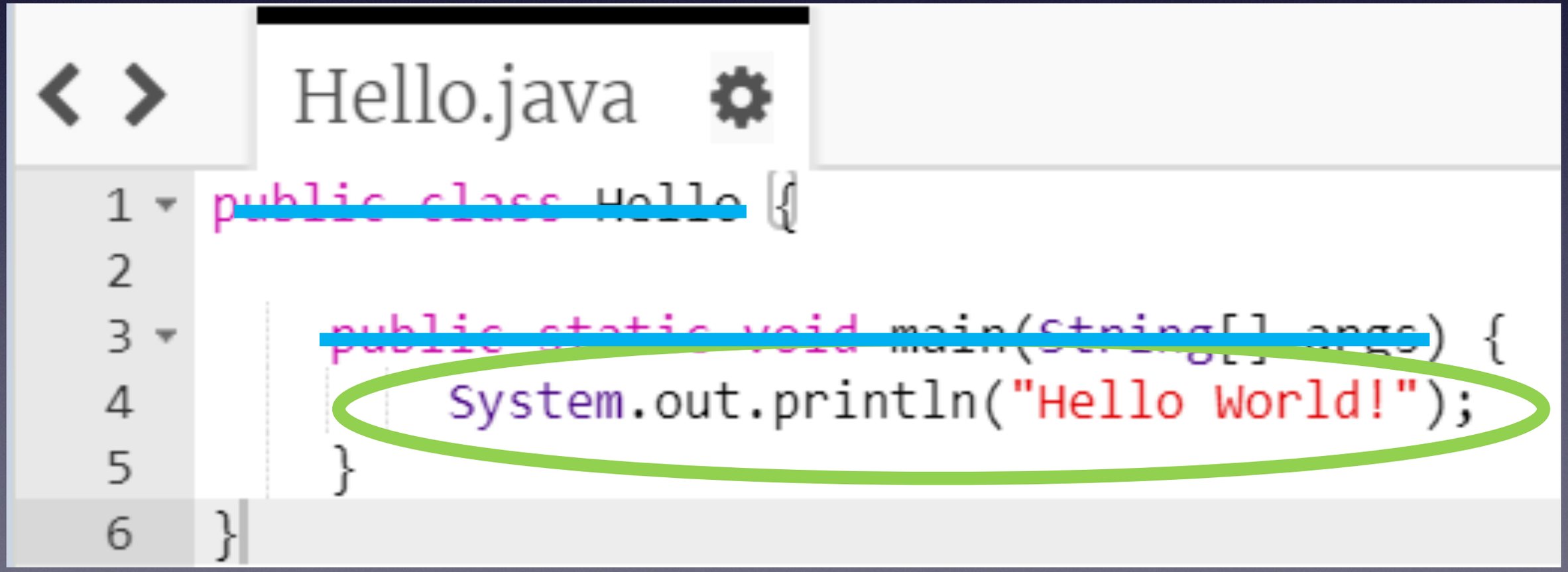
- Click on the link to the “Four 4s Challenge” of just go to CP Java Website
- It might look complicated at first, but . . .

Hello.java

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

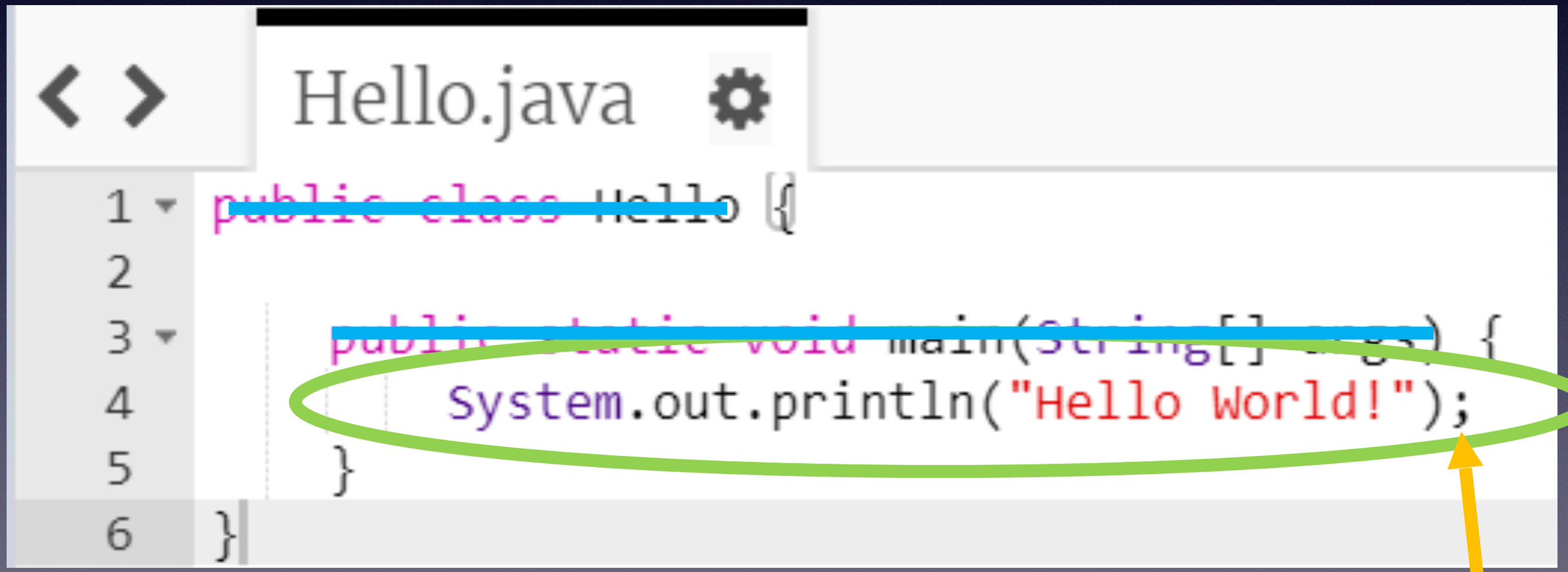
A basic Java program

- It turns out that this line is much more important than the others
- So we can ignore everything except the code circled in green



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`

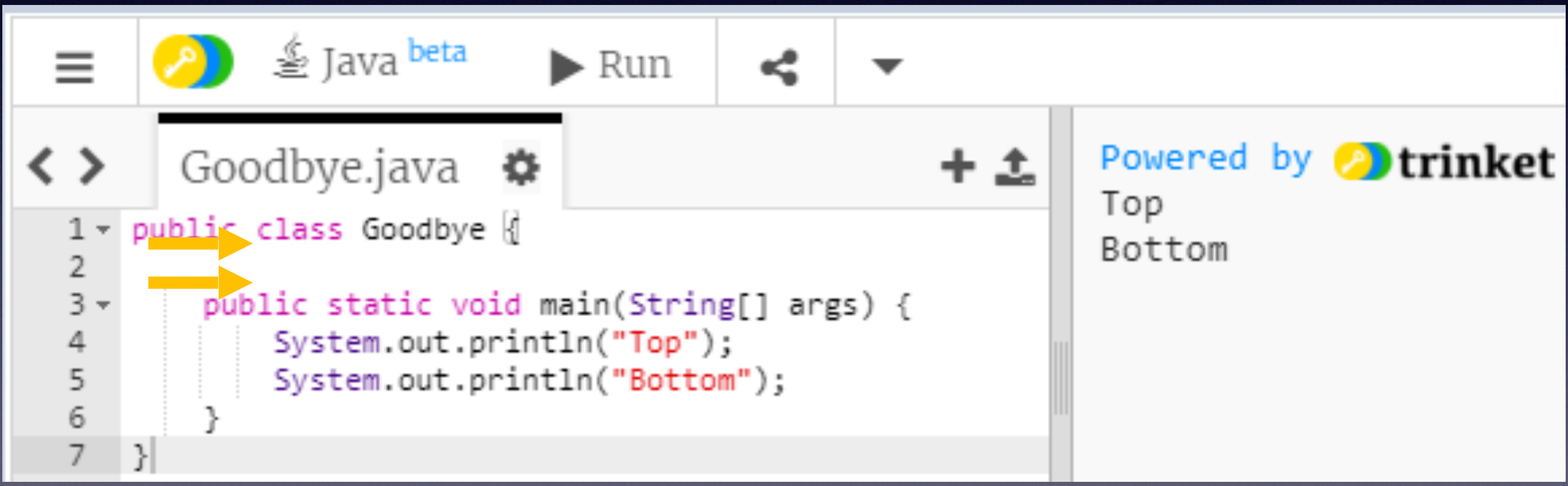
- **`"Hello World!"`** is a Java **`String`**
- A **`String`** is a collection of letters, digits, punctuation and/or spaces
- The beginning and end of the **`String`** are marked with double quotes (**`"`**)

Hello.java

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

`print()` VS `println()`

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

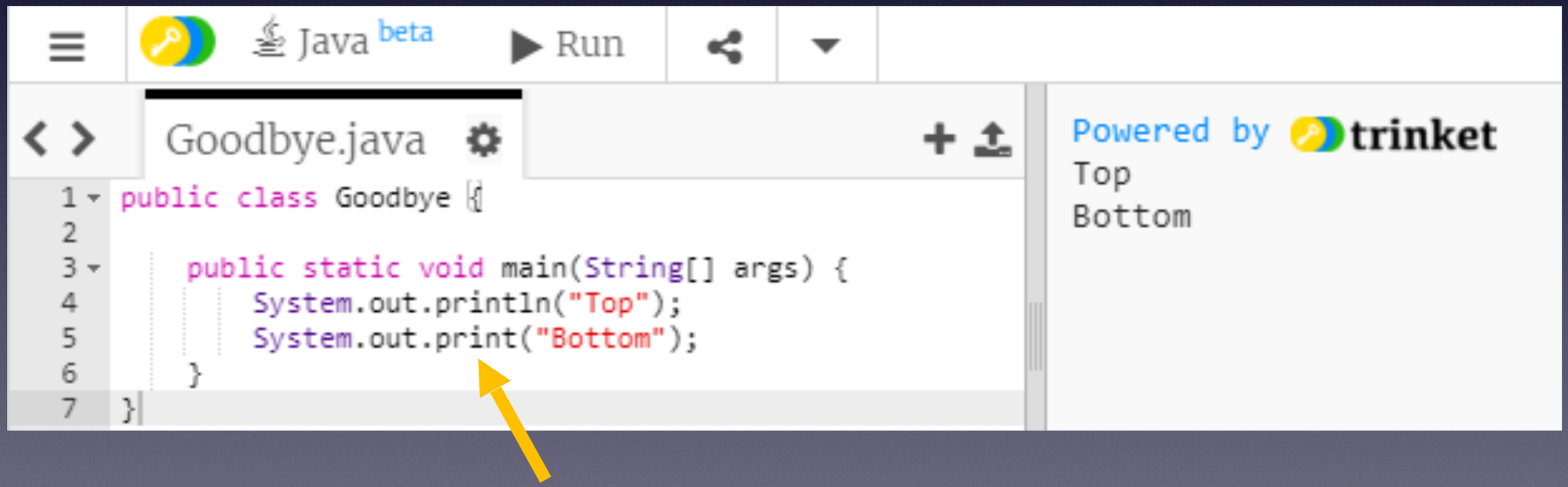


`print()` VS `println()`

- `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"`

`print()` VS `println()`

- Changing the second statement to `System.out.print()` doesn't change the output since nothing is printed after `"Bottom"`



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

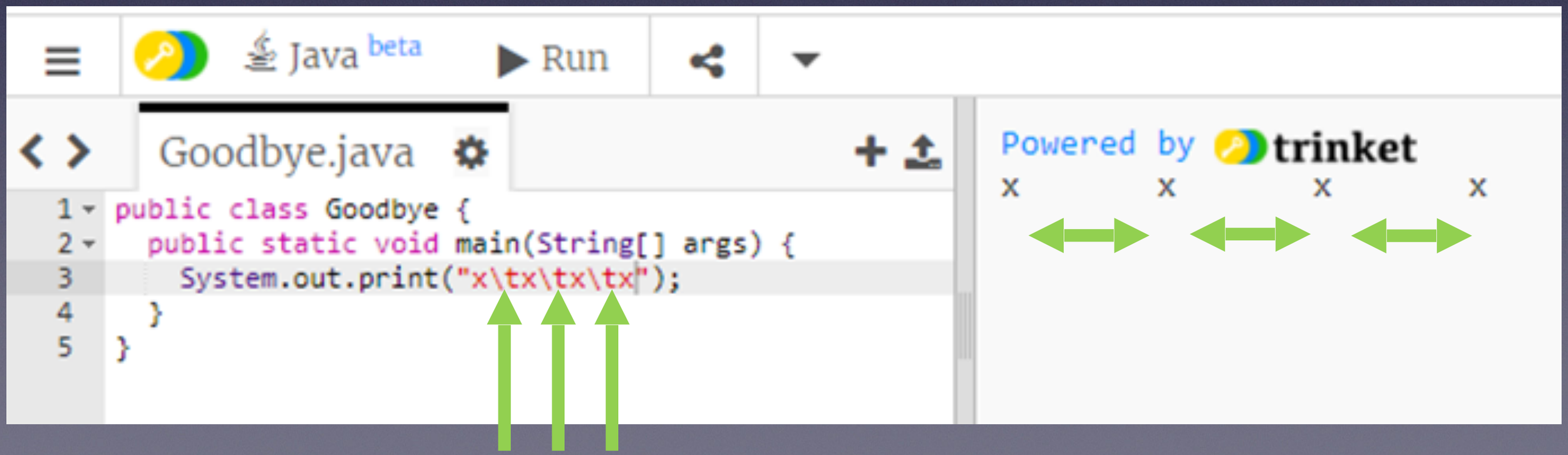
- Special characters
- In Java, Escape Sequences begin with a backslash \
- *A good way to remember the difference between a backslash and a forward slash is that a backslash leans backwards (\ ), while a forward slash leans forward (/)*

Common Java Escape Sequences

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

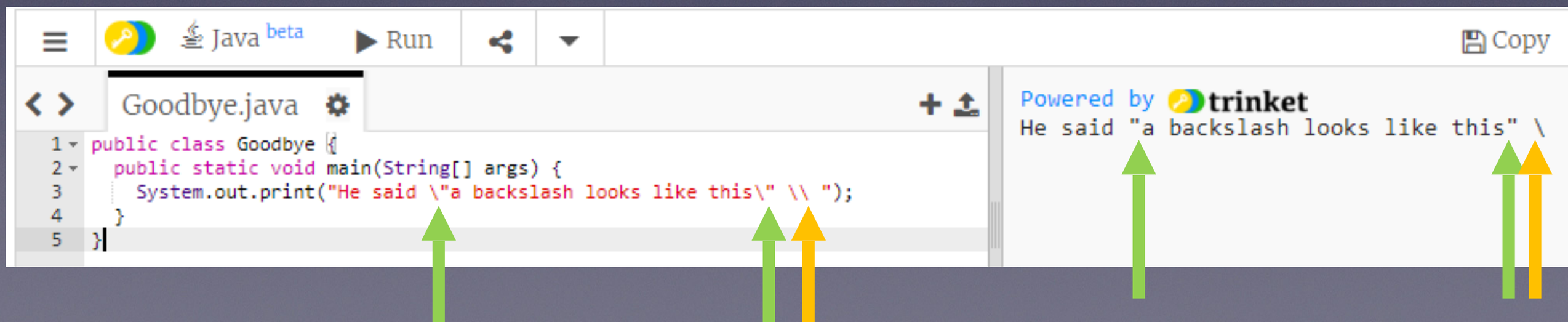
Common Java Escape Sequences

- `\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



Formatting Java code

- In Java programs, some spaces are required
- For example, you need at least one space between keywords
- The program below is not legal

```
publicclassGoodbye{

    publicstaticvoidmain(String[] args) {
        System.out.print("Goodbye, ");
        System.out.println("cruel world");
    }
}
```

Formatting Java code

- But most other spaces are optional
- For example, this program is legal but hard to read

```
public class Goodbye { public static void main(String[] args)
{ System.out.print("Goodbye, "); System.out.println
("cruel world");}}
```

Formatting for easier reading

- The blank space (also called "white space") commonly used to format code is:
  - Indentation inside of { }
  - One statement per line

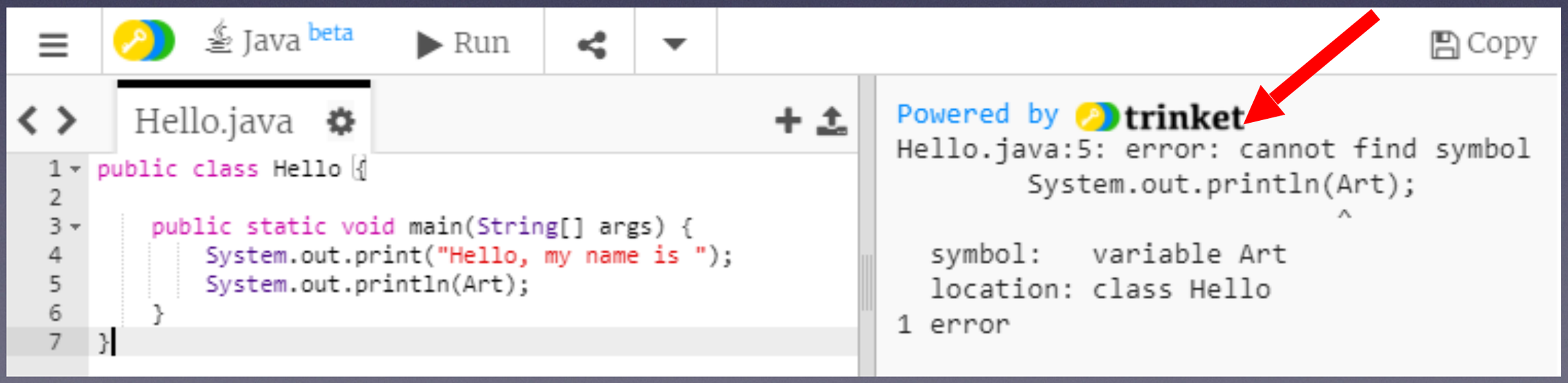
```
public class Goodbye {
    public static void main(String[] args) {
        System.out.print("Goodbye, ");
        System.out.println("cruel world");
    }
}
```

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

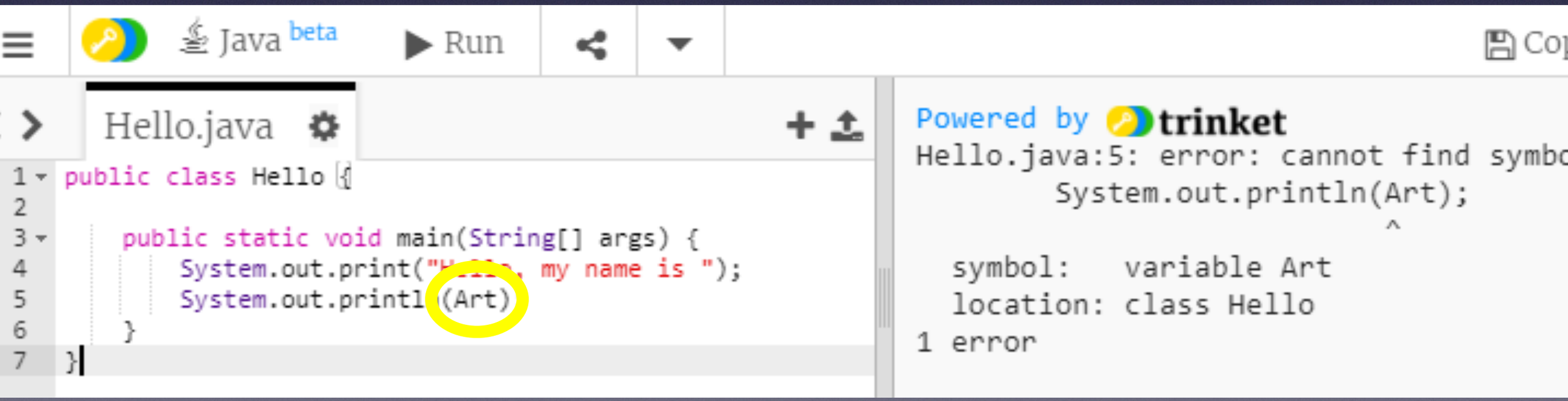
Errors

- 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



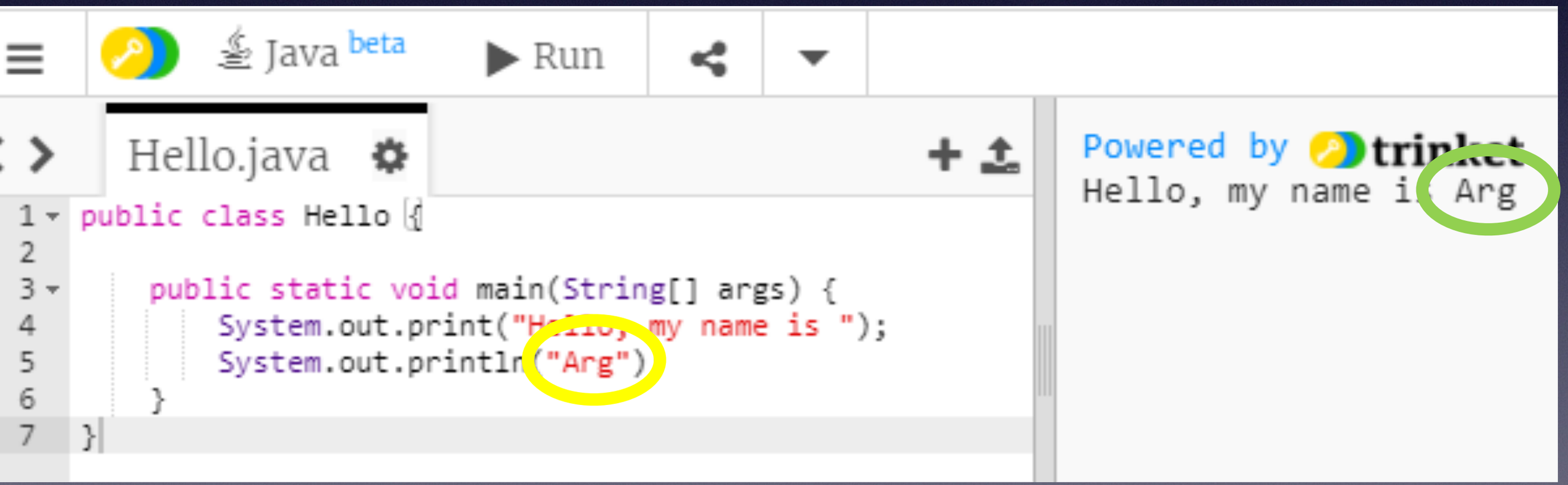
Syntax error

- 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



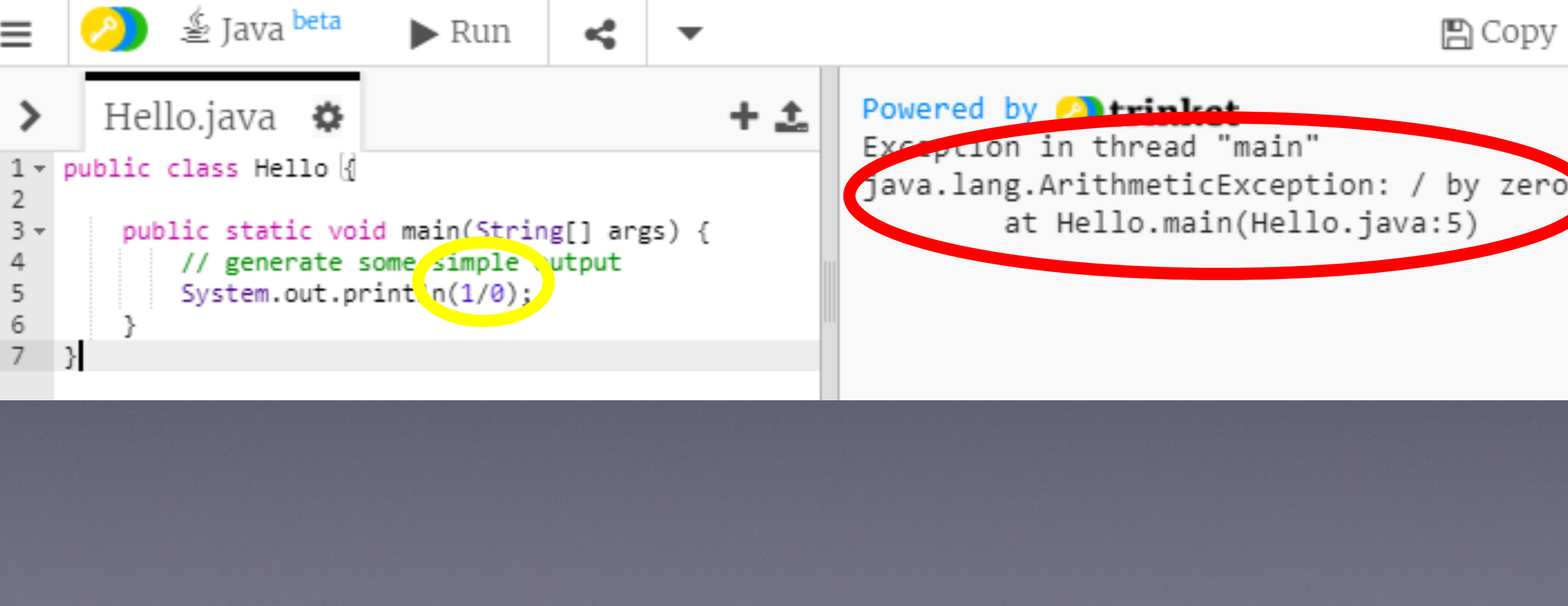
Logic error

- 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

\+ – * /

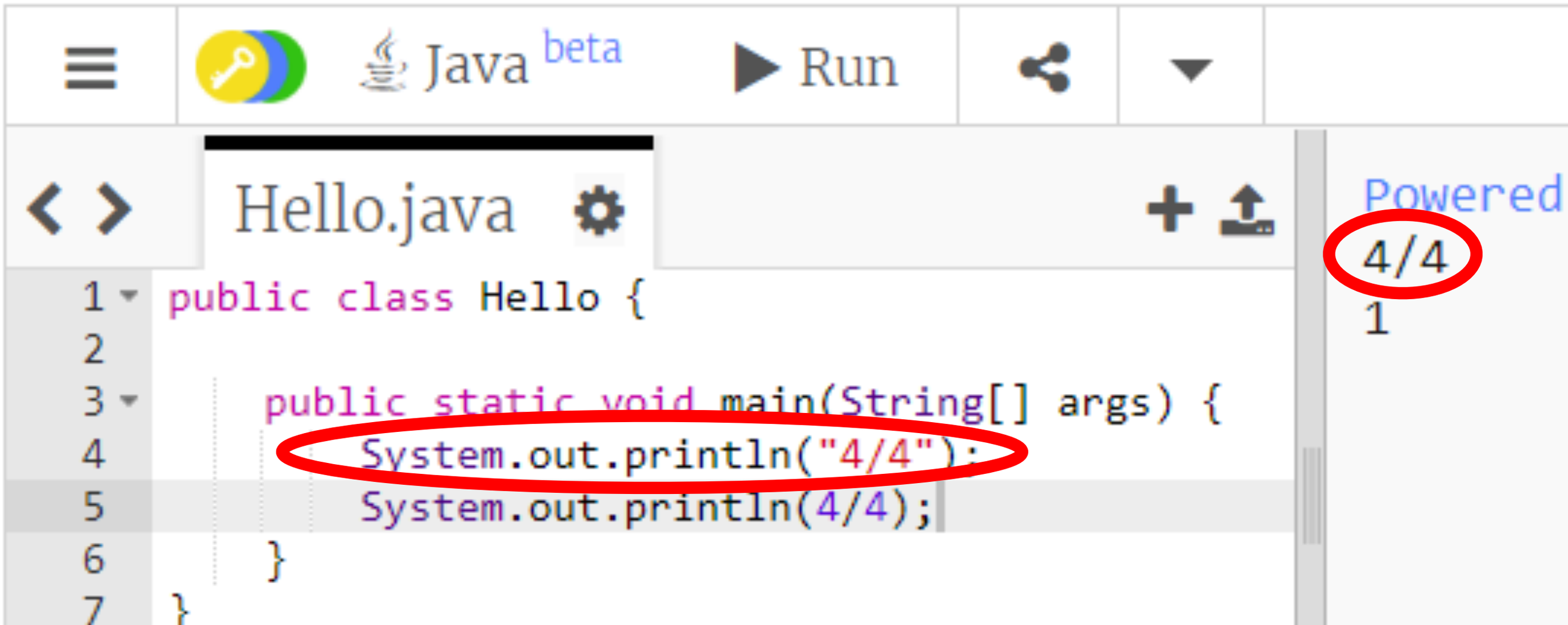
- Addition
- Subtraction
- Multiplication
- Division

Literals vs. Expressions

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

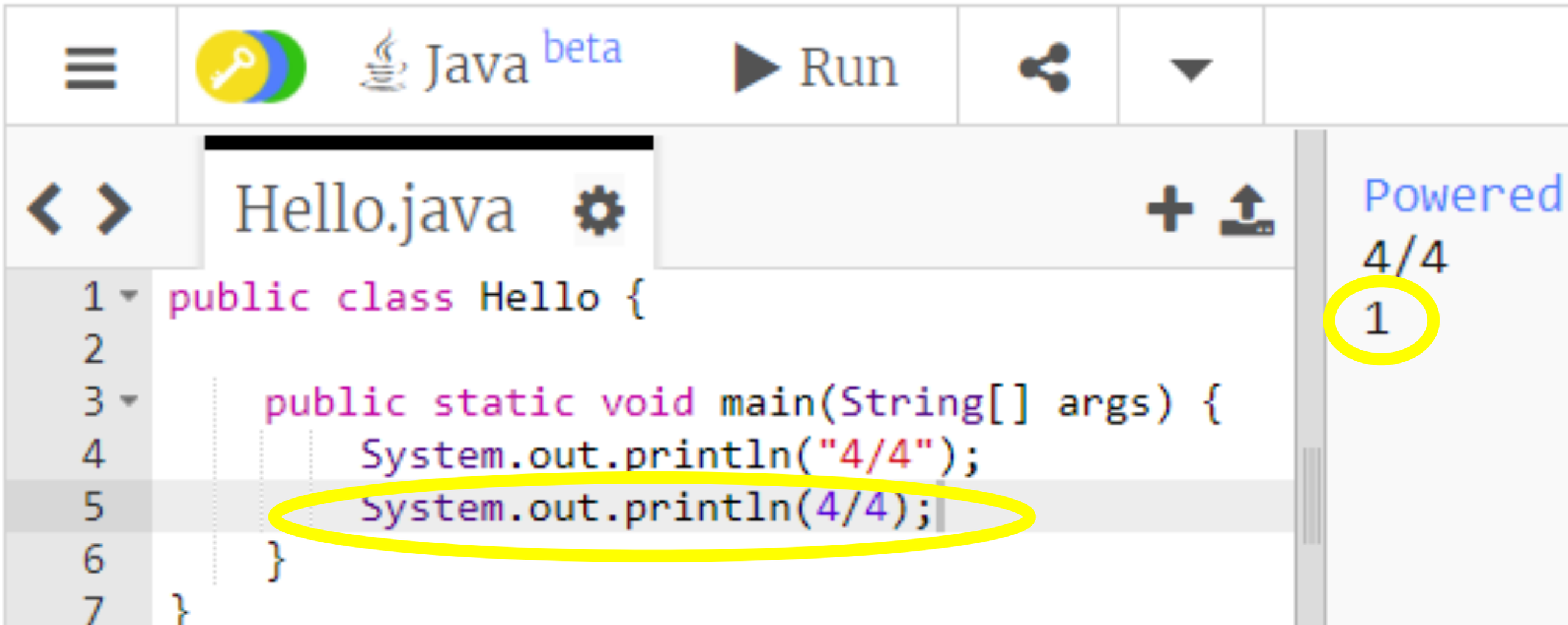
Literals vs. Expressions

- 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

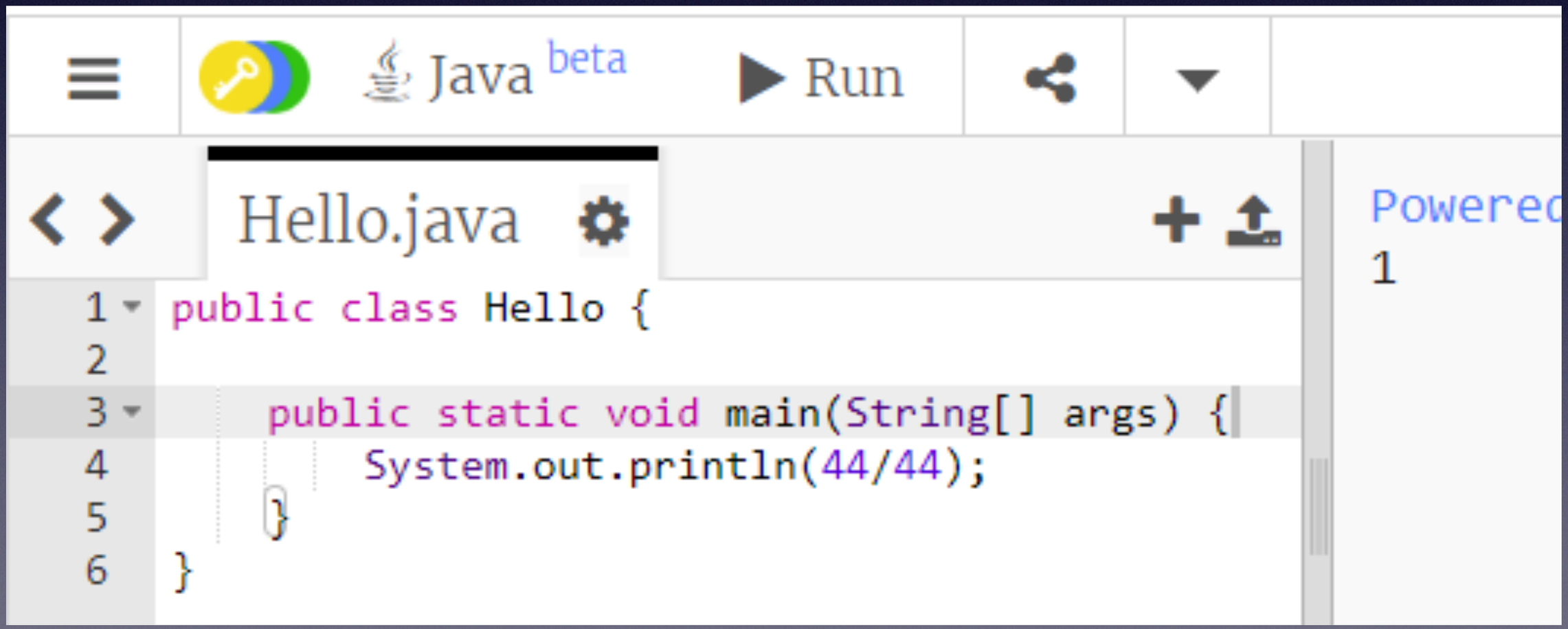


Four 4s challenge

- 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.

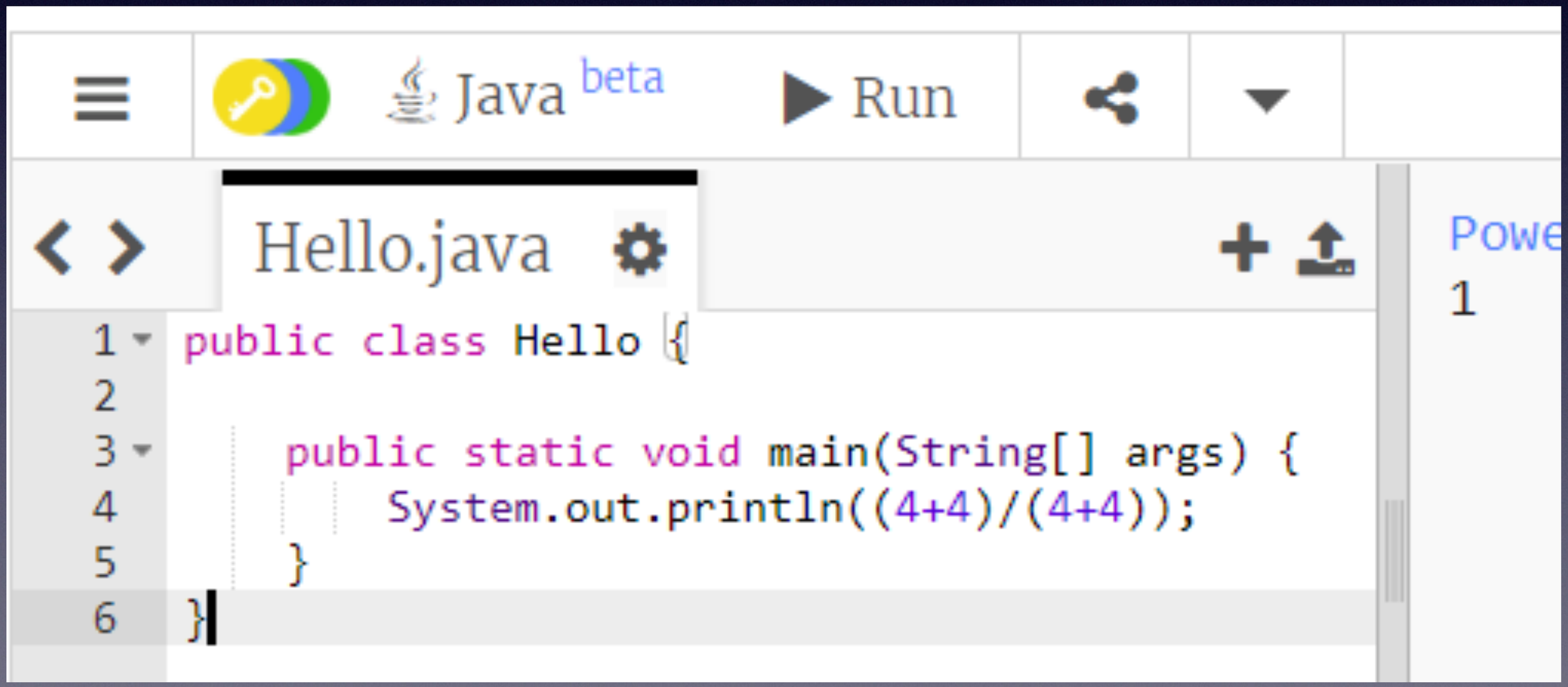
Four 4s challenge

- 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.



Java Intro

Basics Lesson
Chapter 2 - Variable Types Integer & Modulo Division Strings

Chapter 2

- 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(123123123123123123123123);
6      }
7  }
```

Copy

Powered by trinket

```
Hello.java:5: error: integer number too large:
123123123123123123123123

System.out.println(123123123123123123123123);
                   ^

1 error
```

Primitive data types

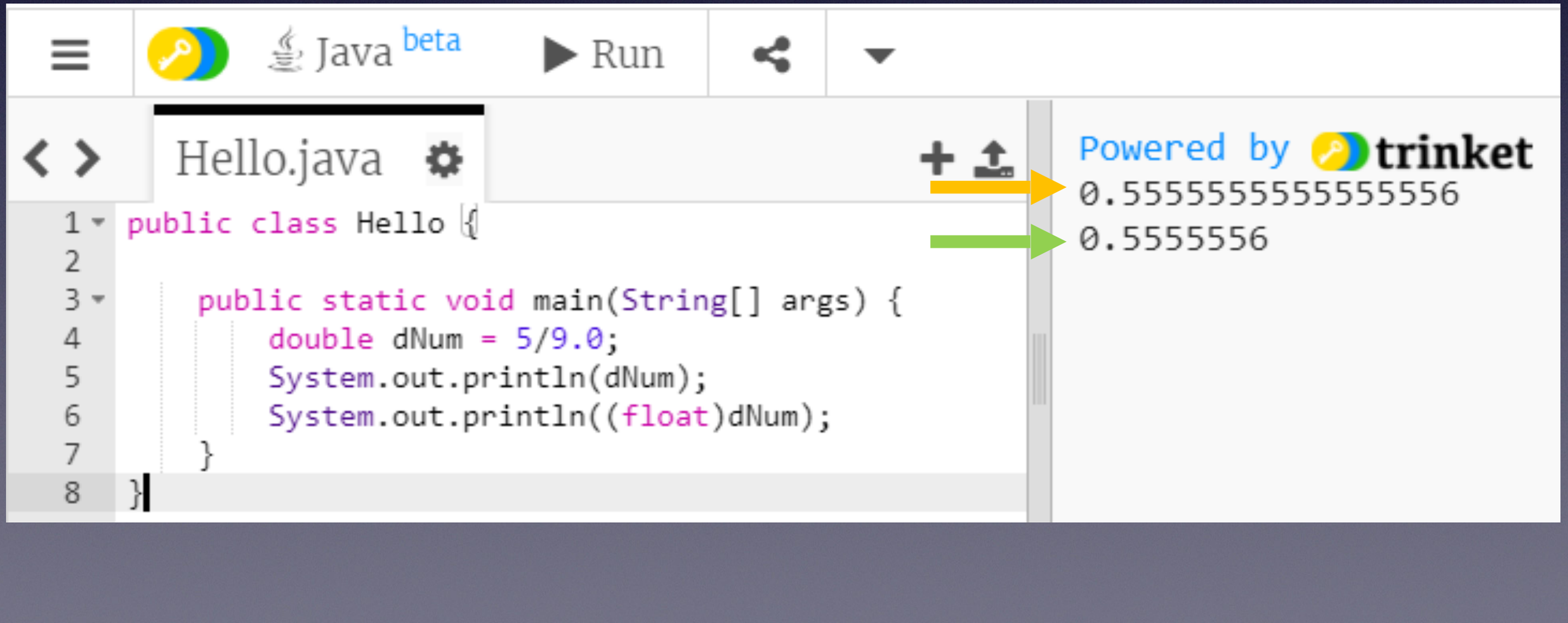
- In Java (unlike Python or JavaScript) variables have ***types***
- Each type has size limits
- For this class, you need to know 5 basic (aka "primitive") types:
  - **`int`**
  - **`float`**
  - **`double`**
  - **`boolean`**
  - **`char`**

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'`**

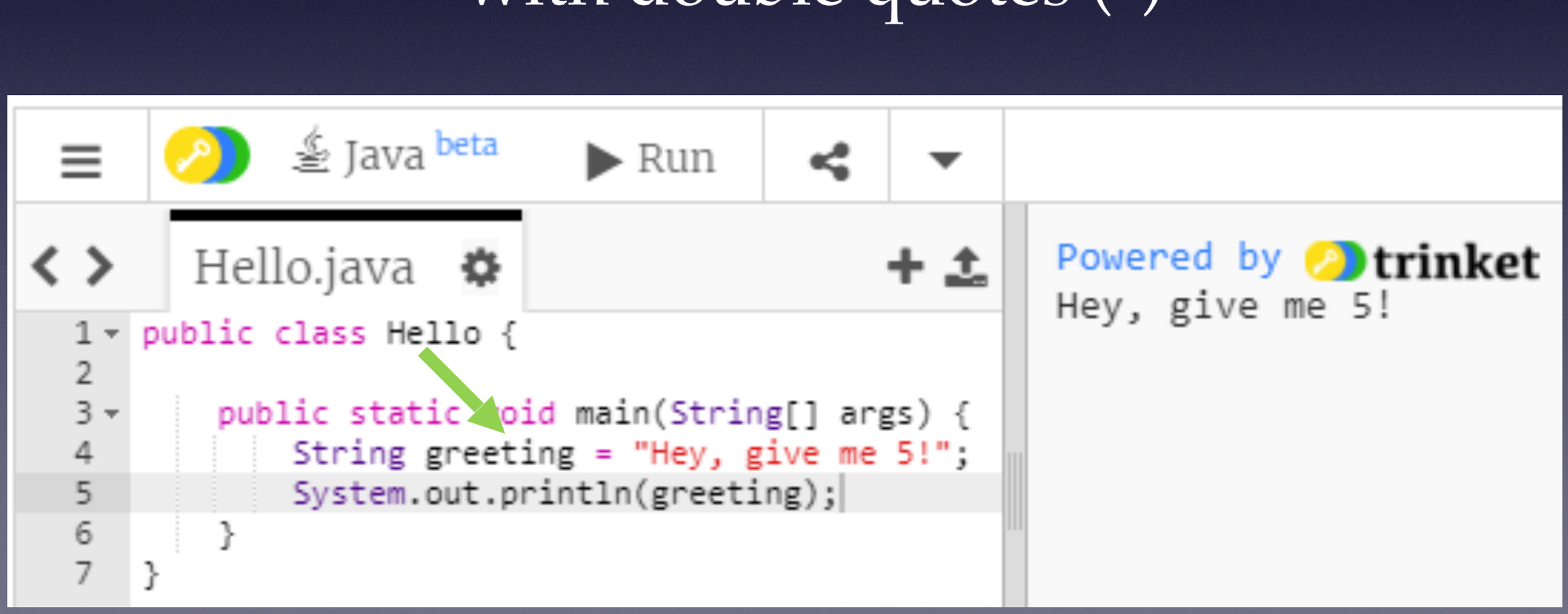
`float` vs. `double`

- `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`



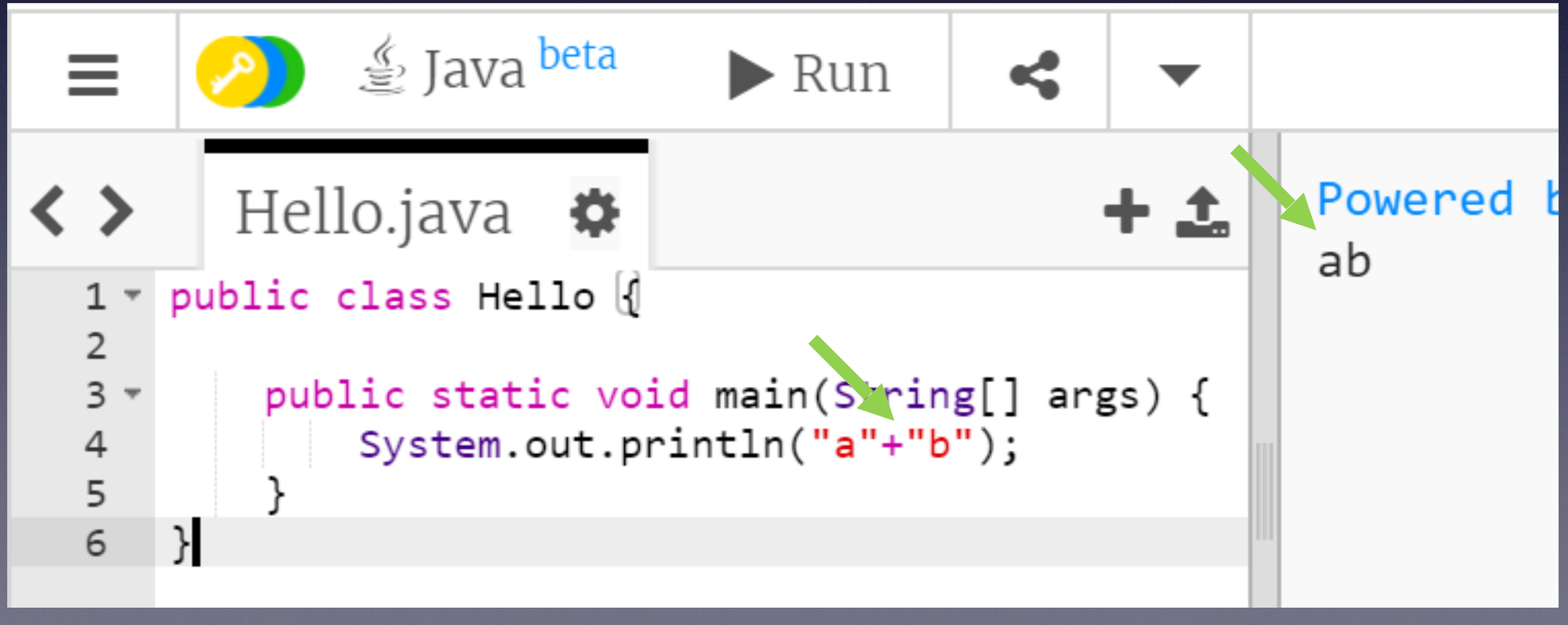
`String` variables

- 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`

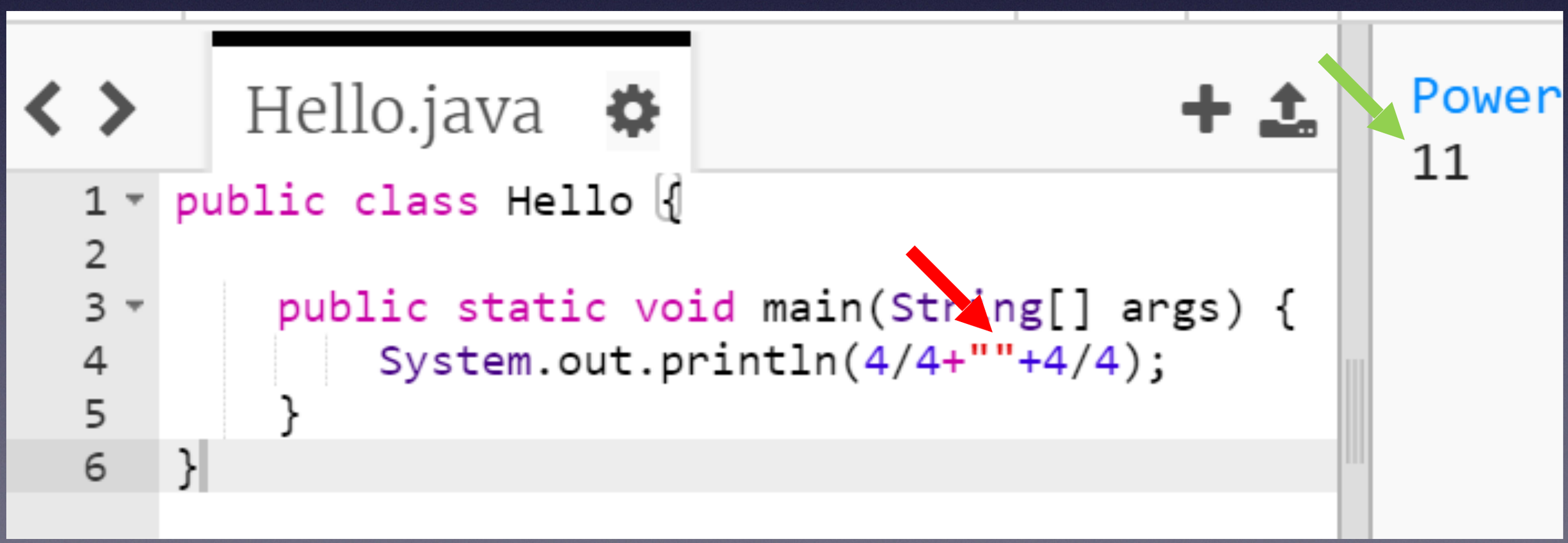
- Java executes from left to right so, `1 + 2` is `3`, and `3 + "Hello"` is `"3Hello"`
- `"Hello" + 1` is `"Hello1"`, and `"Hello1" + 2` is `"Hello12"`

```
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

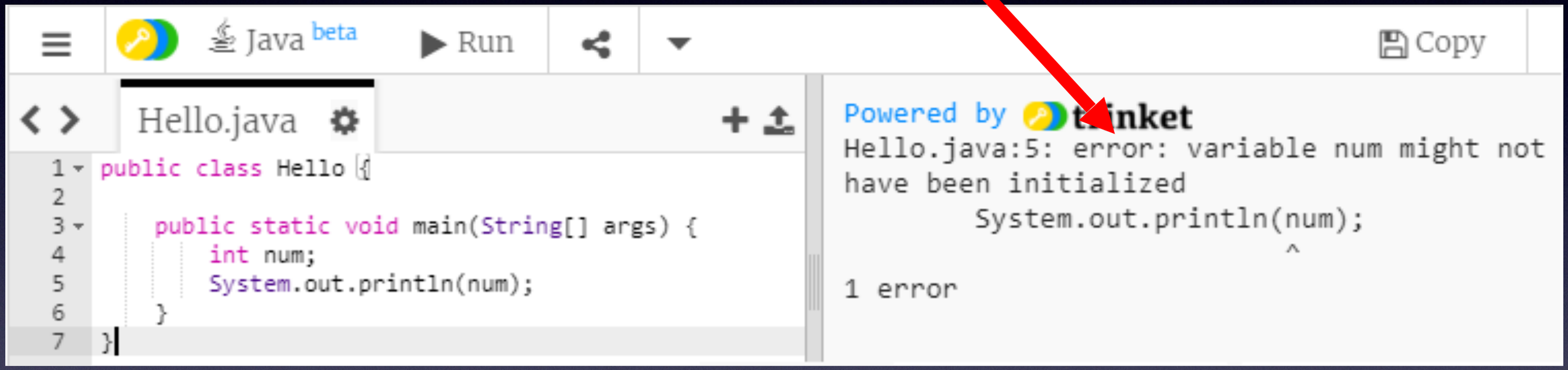
Variable names and keywords

- 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

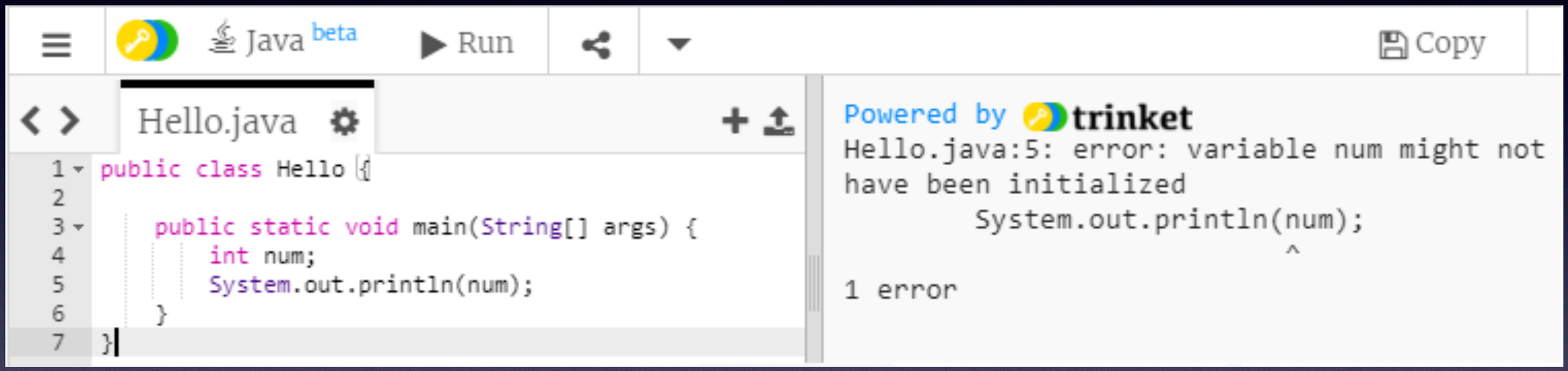
- Because a Java variable name can't have spaces, a style called **camelCase** is usually used for variable names with more than one word
- **camelCase** capitalizes the first letter of each word except the first word
- Examples: **`firstName`**, **`numberOfDogs`**
- Java variable names are case-sensitive,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

Arithmetic operators

\+ − * / %

- Addition
- Subtraction
- Multiplication
- Division
- Modulus (also *Mod* or *Modulo*) calculates the remainder of dividing two integers

Modulus and integer division

Remember how you did math in grade school?

$$5 \overline{)\,8}$$

Modulus and integer division

Remember how you did math in grade school?

$$5\overline{)\,8}\qquad 1$$

Modulus and integer division

Remember how you did math in grade school?

$$
\begin{array}{r}
1 \\
5\overline{)\,8} \\
5 \\
\hline
\end{array}
$$

Modulus and integer division

Remember how you did math in grade school?

$$
\begin{array}{r}
1 \\
5\overline{)\,8} \\
\underline{5} \\
3
\end{array}
$$

Modulus and integer division

The modulus operator gives the remainder of an integer division expression

8 / 5

8 % 5

$$\begin{array}{r} 1 \\ 5\overline{)\,8} \\ \underline{5} \\ 3 \end{array}$$

Kahoot!

Integer Division in Java