

Java Intro

Basics Lesson

Chapter 1 - Variables Loops Functions Branching

Java Basics

The common building blocks in programming languages
are:

Variables

Loops

if statements

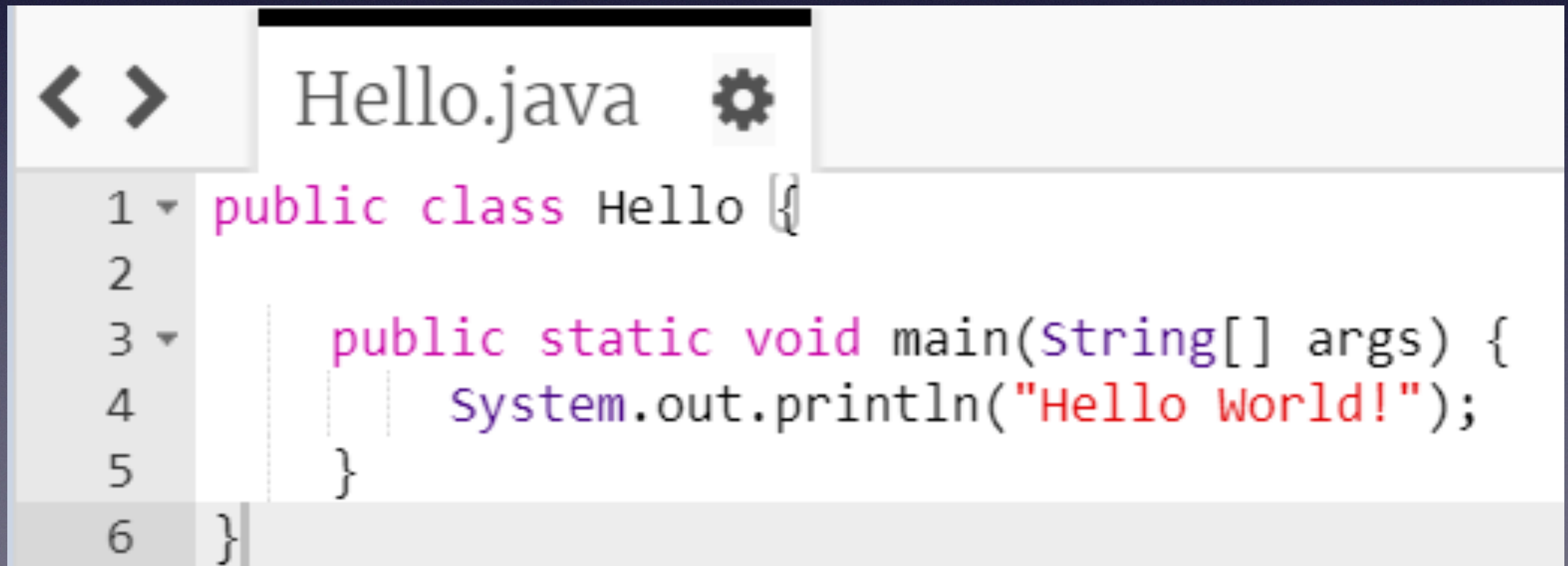
Functions (aka “methods”)

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 . . .

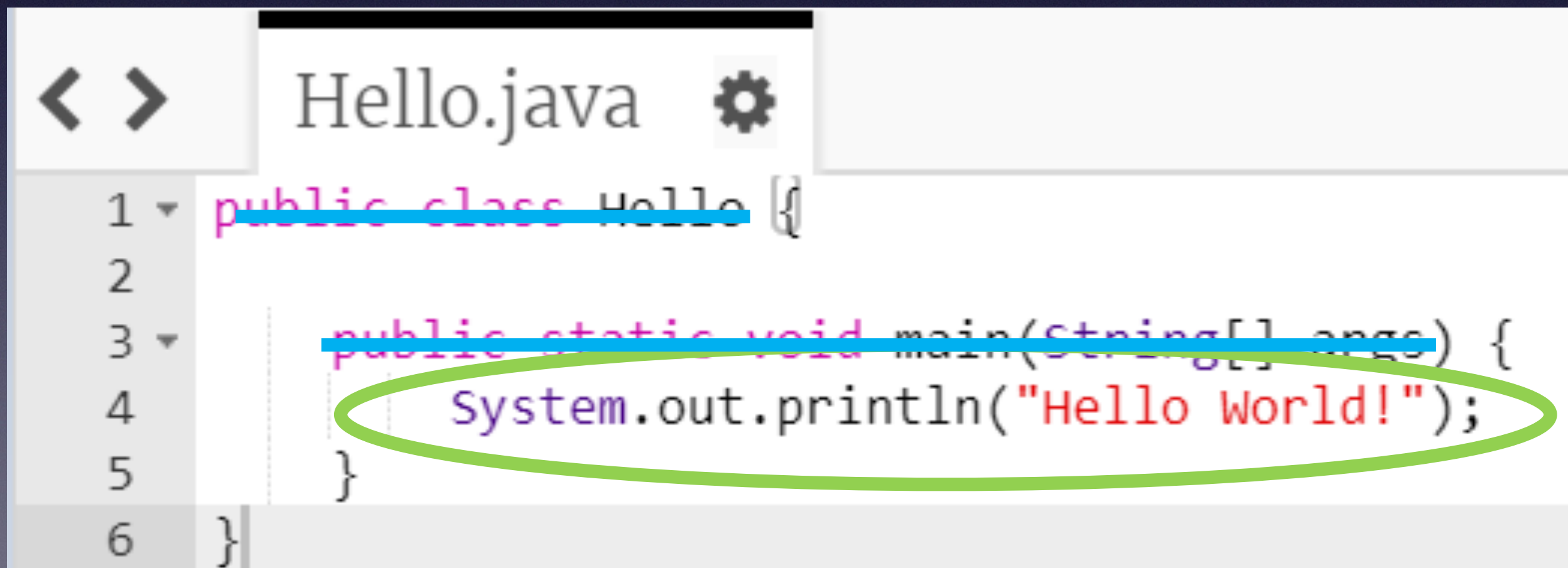


The screenshot shows a code editor window titled "Hello.java" with a gear icon for settings. The code is as follows:

```
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**



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

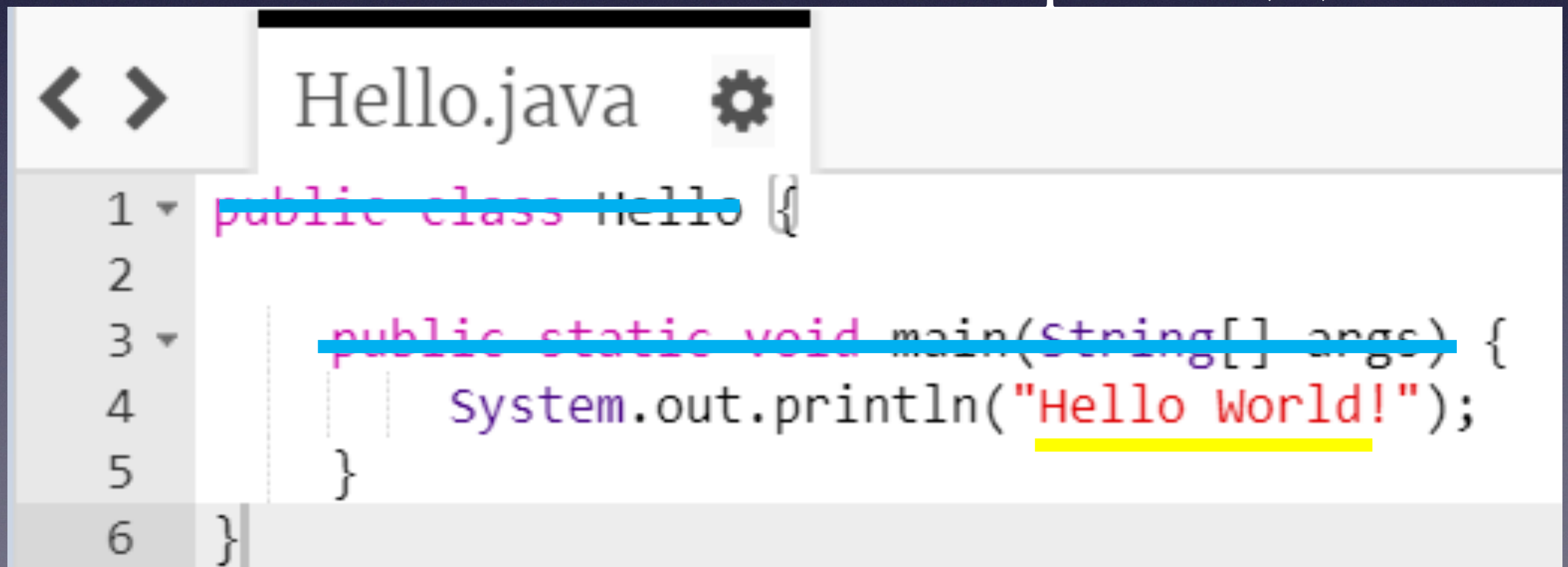

Statements

- 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

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


String

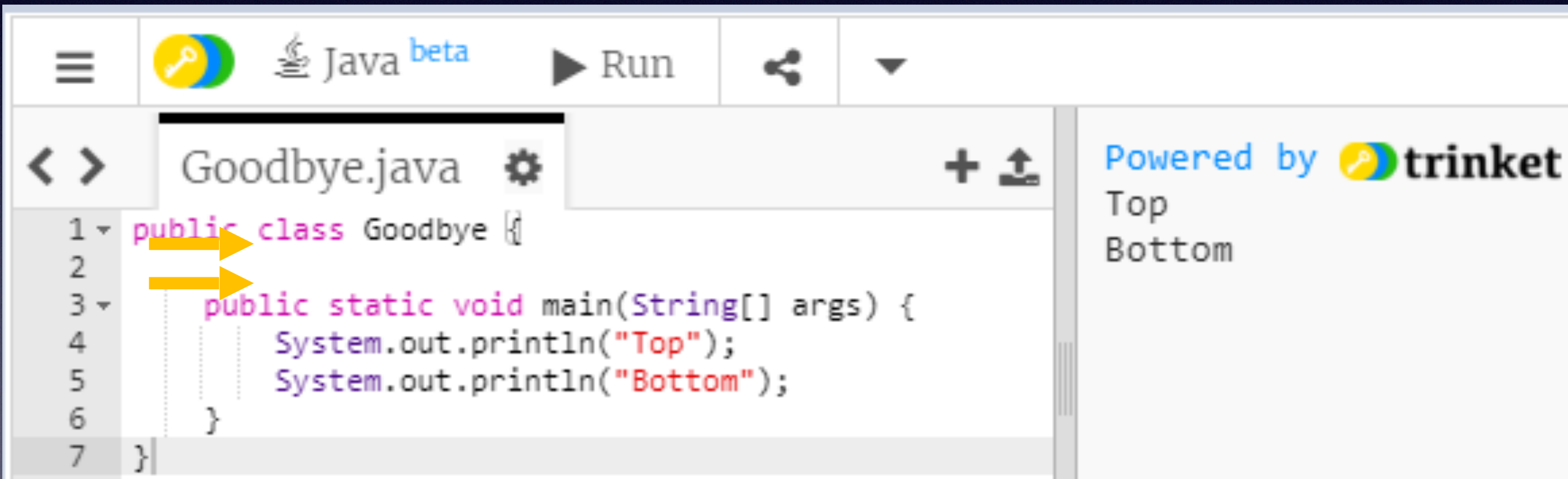
- **"Hello World!"** is a Java **String**
- A **String** is a collection of letters, digits, punctuation and / or spaces
- The beginning and end of the **String** are marked with double quotes (")



```
< > 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




The screenshot shows a Java IDE interface. At the top, there's a toolbar with icons for a menu, a key, a Java logo with 'beta' text, a 'Run' button, a share icon, and a dropdown arrow. Below the toolbar, the file name 'Goodbye.java' is displayed with a settings gear icon and a plus icon. The main editor area contains the following Java code:

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

Two yellow arrows point from the left margin to the opening curly brace of the class and the opening curly brace of the main method. On the right side of the IDE, there's a panel that says 'Powered by trinket' with a key icon, and below it, the output of the program is shown as 'Top' followed by 'Bottom' on a new 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"



The screenshot shows a Java IDE interface. The top bar includes a menu icon, a key icon, the text "Java beta", a "Run" button, and a share icon. The editor window displays the file "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 }
```

An orange arrow points to the `System.out.print("Top");` line. To the right of the editor, the output is displayed: "Powered by trinket" followed by "TopBottom" on the same line, with "TopBottom" underlined in green.

print() vs println()

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



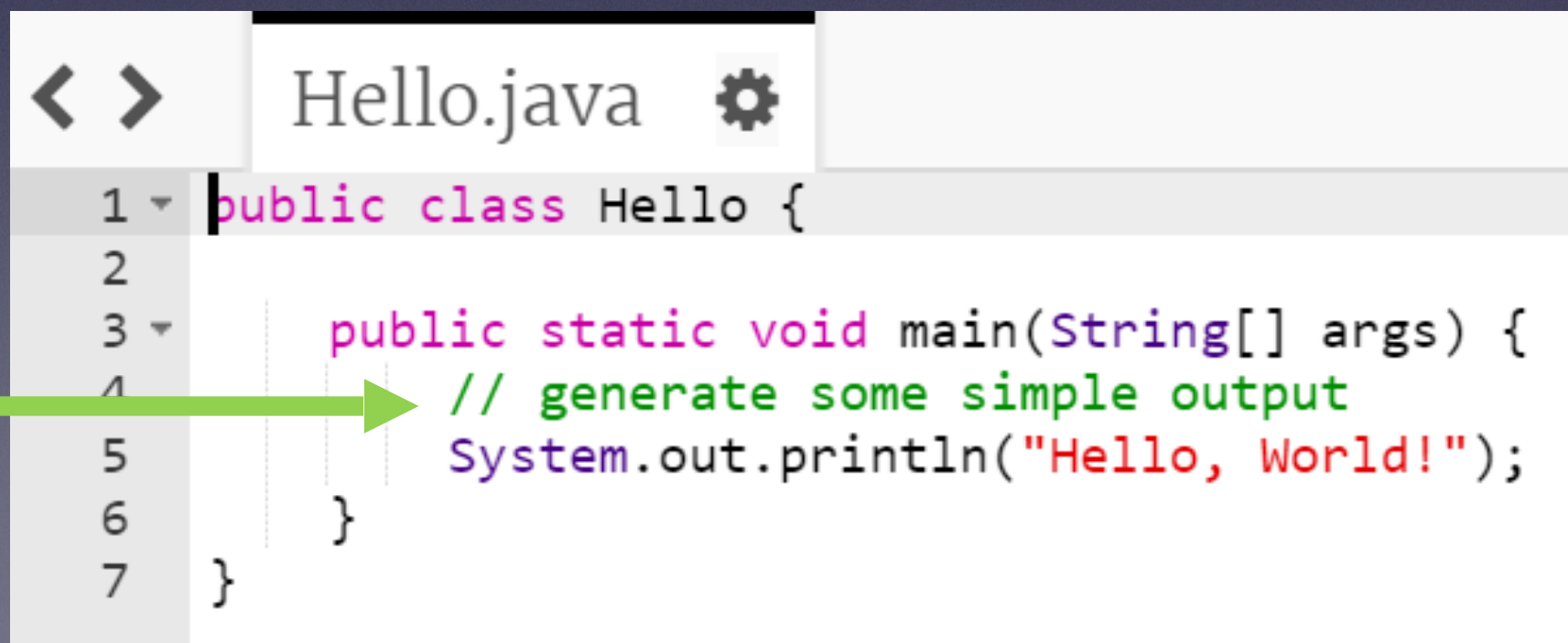
The screenshot shows a Java IDE interface. At the top, there's a toolbar with icons for a menu, a key, a Java logo with 'beta', a 'Run' button, a share icon, and a dropdown arrow. Below the toolbar, the file 'Goodbye.java' is open. The code in the editor is as follows:

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

A yellow arrow points to the `System.out.print("Bottom");` line on line 5. To the right of the code editor, the output is displayed: 'Top' followed by 'Bottom' on the next line. The output is powered by 'trinket'.

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



The screenshot shows a code editor window titled "Hello.java" with a gear icon for settings. The code is as follows:

```
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 }
```

A green arrow points from the left margin to the comment on line 4: `// generate some simple output`.

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



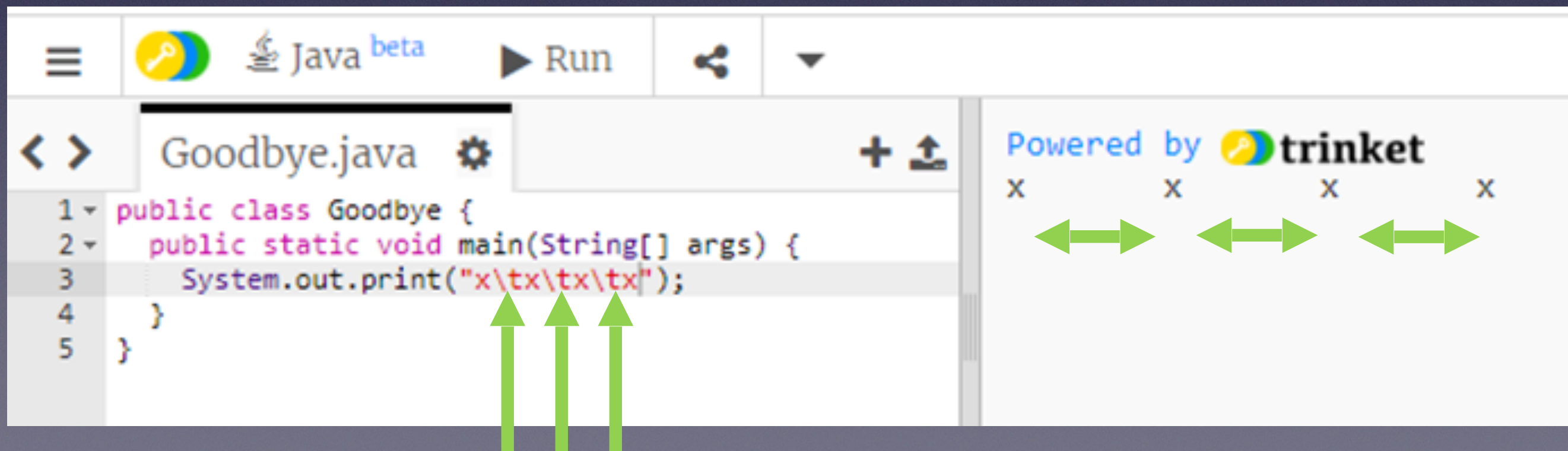
The screenshot shows a Java IDE interface. At the top, there's a toolbar with a menu icon, a key icon, a Java logo with 'beta' text, a blue 'Run' button, and a share icon. Below the toolbar, the file name 'Goodbye.java' is displayed with a gear icon and a plus icon. The code editor shows the following code:

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

A green arrow points upwards to the `\n` escape sequence in the string "Top\nBottom" on line 3. To the right of the code editor, there's a panel that says 'Powered by trinket' with a key icon, and below it, the text 'Top' and 'Bottom' is displayed, demonstrating the effect of the newline escape sequence.

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

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

Powered by trinket

He said "a backslash looks like this" \

The screenshot shows a Java IDE with a file named 'Goodbye.java'. The code defines a class 'Goodbye' with a 'main' method that prints the string "He said \"a backslash looks like this\" \\ ". Three arrows point from the code to the output: a green arrow from the opening double quote, a green arrow from the escaped double quote, and an orange arrow from the escaped backslash. The output on the right shows the result: "He said \"a backslash looks like this\" \".

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**

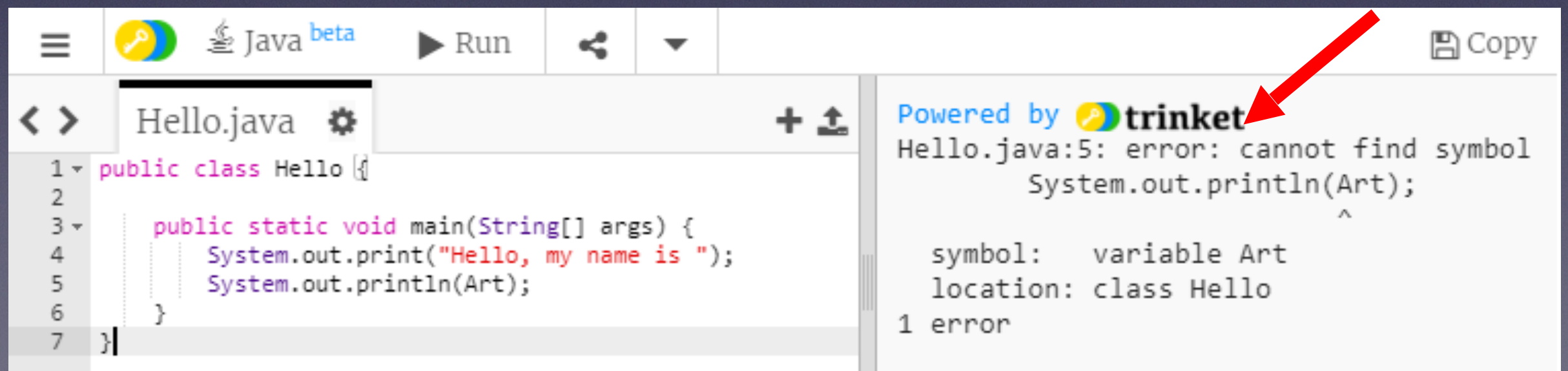
```
public class Goodbye {  
    public static void main(String[] args) {  
        System.out.print("Goodbye, ");  
        System.out.println("cruel world");  
    }  
}
```


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



The screenshot shows a Java IDE interface. The top bar includes a menu icon, a key icon, the text "Java beta", a "Run" button, and a "Copy" button. The main editor area displays a file named "Hello.java" with the following code:

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

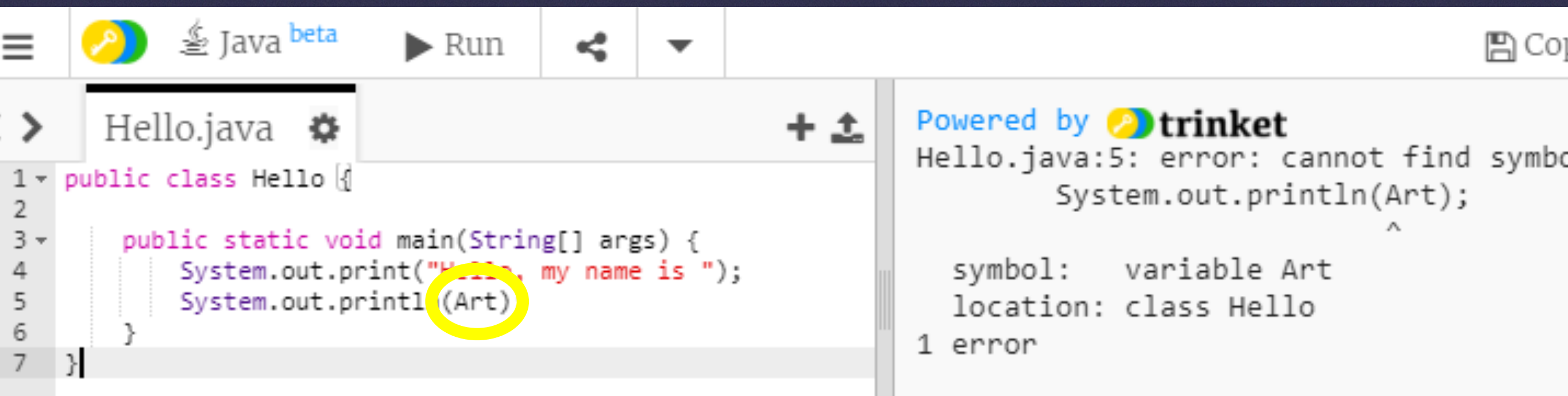
The code on line 5 contains a compilation error. The error message, displayed in the right-hand pane, is:

```
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 top right of the error message pane to the "trinket" logo in the "Powered by" 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




```
public class Hello {  
    public static void main(String[] args) {  
        System.out.print("Hello, my name is ");  
        System.out.println(Art);  
    }  
}
```

Powered by  trinket

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



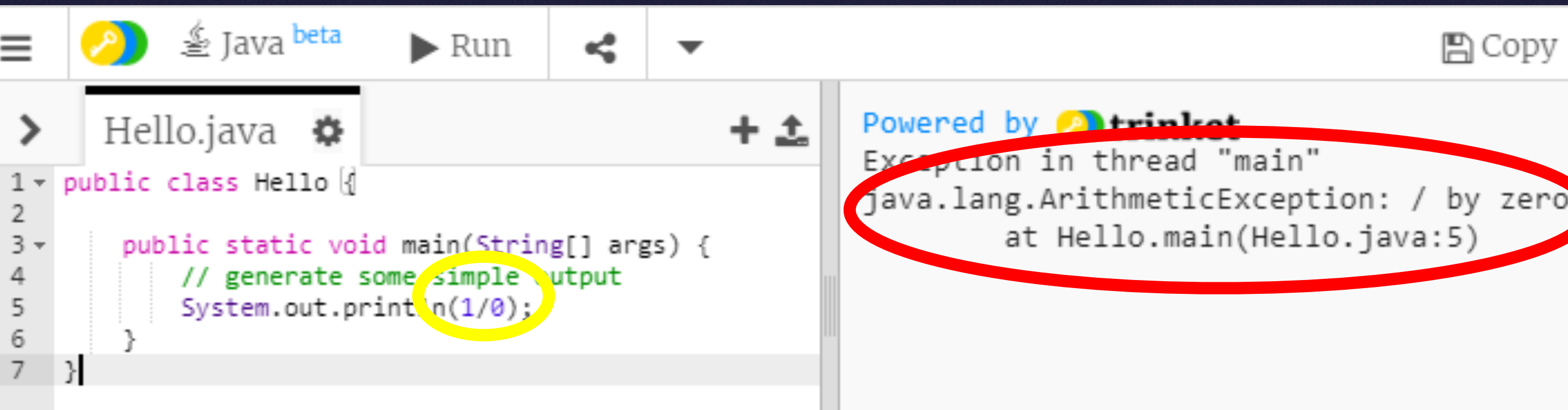
The screenshot shows a Java IDE interface. The top toolbar includes a menu icon, a key icon, a Java logo with 'beta' text, a 'Run' button with a play icon, a share icon, and a dropdown arrow. The editor window is titled 'Hello.java' and contains the following code:

```
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 }
```

The word 'Arg' in the `println` statement on line 5 is circled in yellow. To the right of the editor, the output window shows the text 'Hello, my name is Arg', where 'Arg' is circled in green. The output window also displays 'Powered by trinket'.

(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



The screenshot shows a Java IDE interface. On the left, a code editor displays the following Java code for 'Hello.java':

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

The expression `1/0` in line 5 is circled in yellow. On the right, a console window shows the output of the program, which is a runtime exception:

```
Powered by trinket  
Exception in thread "main"  
java.lang.ArithmeticException: / by zero  
    at Hello.main(Hello.java:5)
```

The entire exception message in the console is circled in red.

Arithmetic in Java

+ - * /

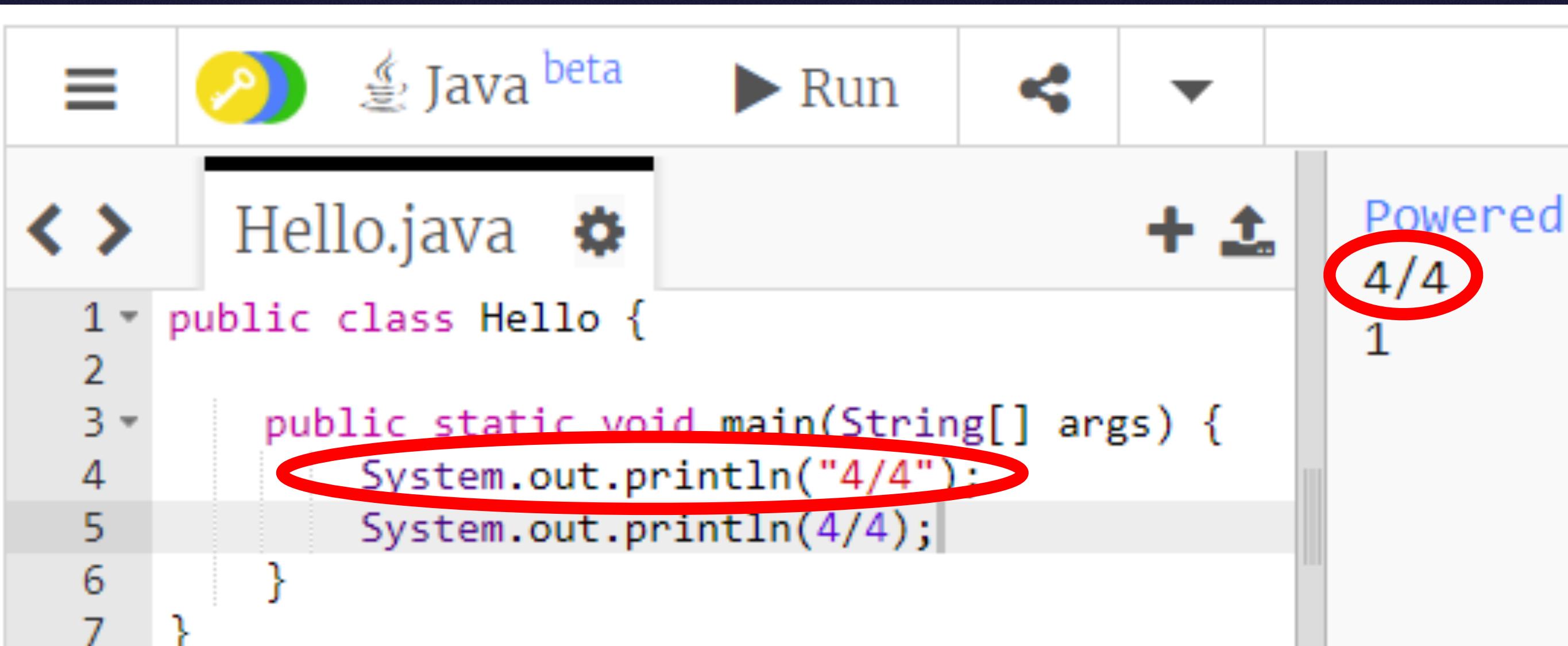
- Addition
- Subtraction
- Multiplication
- Division

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 **prints** it in exactly the same form



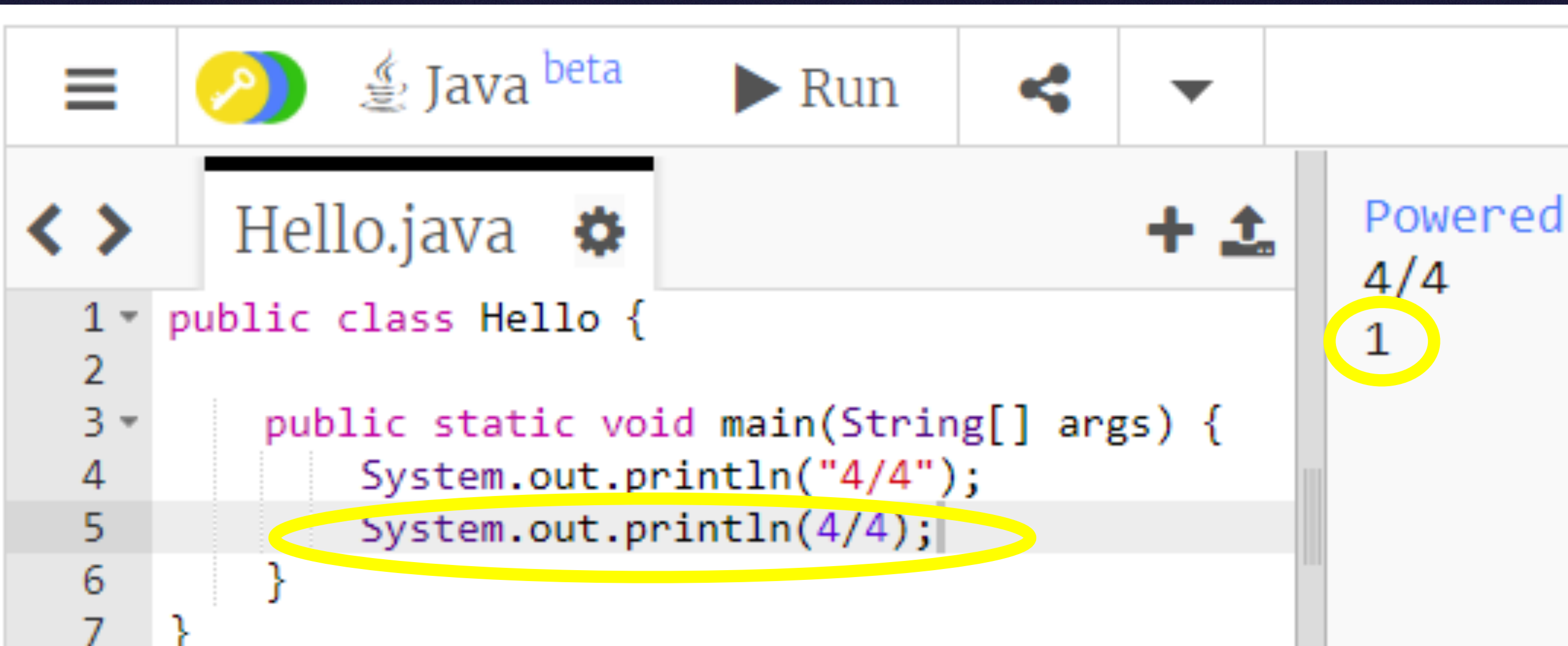
The screenshot shows an IDE interface for Java. The top bar includes a menu icon, a key icon, the text "Java beta", a "Run" button, and a share icon. Below the top bar, the file name "Hello.java" is displayed with a gear icon. The code editor shows the following code:

```
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 code is displayed with line numbers 1 through 7 on the left. The string literal `"4/4"` on line 4 and the numeric expression `4/4` on line 5 are both circled in red. On the right side of the editor, the text "Powered" is visible, and below it, the text `4/4` is circled in red, with the number `1` below it.

Literals vs. Expressions

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



The screenshot shows an IDE window titled "Hello.java" with a "Run" button and a "Java beta" label. The code is as follows:

```
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 }
```

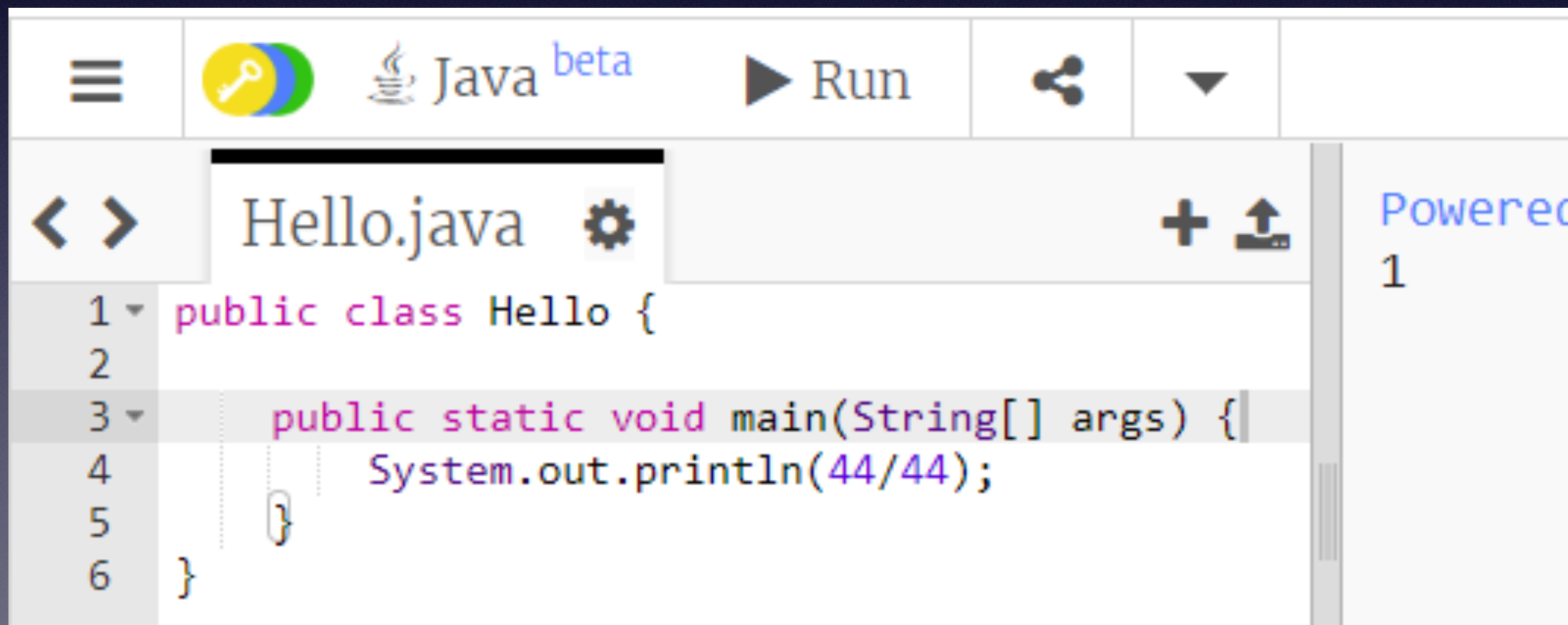
The line `System.out.println(4/4);` is circled in yellow. On the right side of the IDE, the output is displayed as "Powered 4/4" with the result "1" circled in yellow.

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



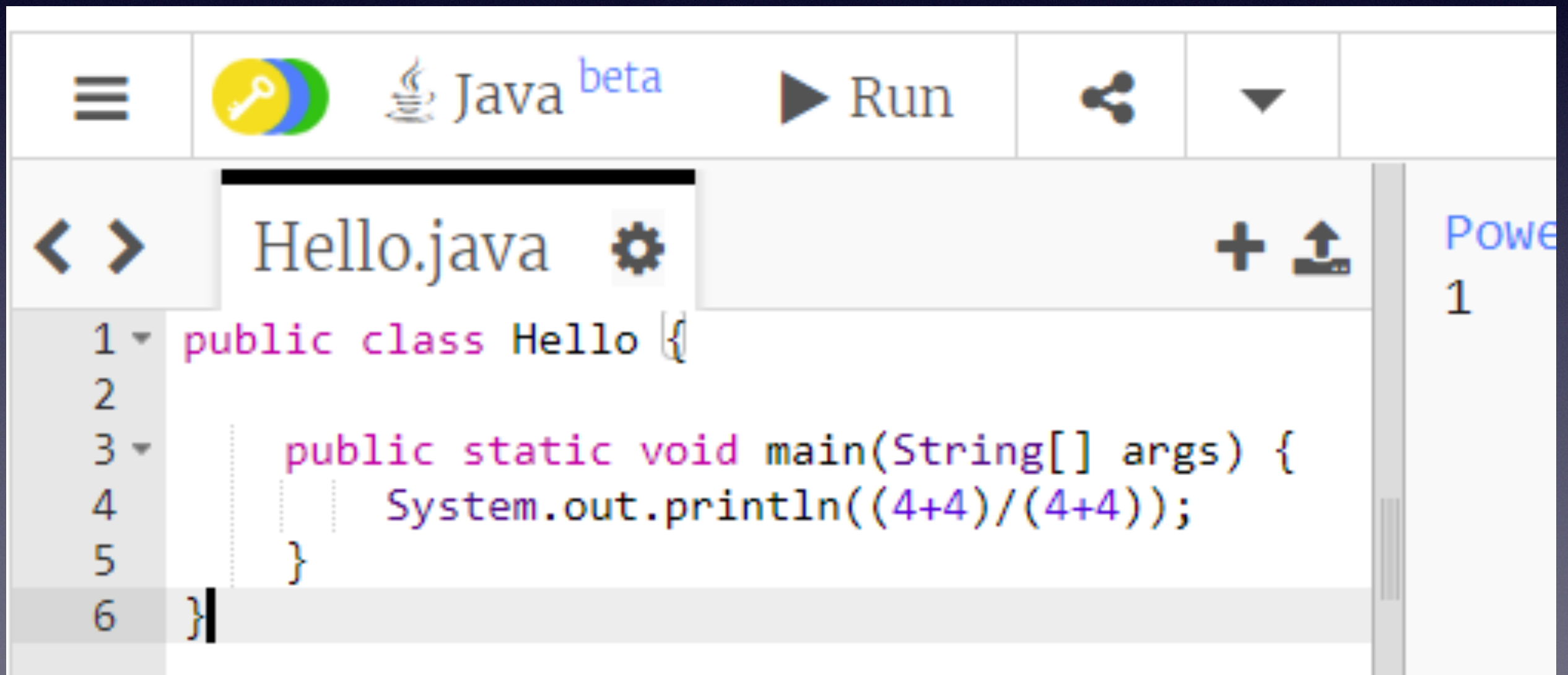
The screenshot shows a Java IDE interface. At the top, there is a toolbar with a menu icon, a key icon, the text "Java beta", a "Run" button with a play icon, a share icon, and a dropdown arrow. Below the toolbar, a tab labeled "Hello.java" with a gear icon is active. The code editor displays the following Java code:

```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         System.out.println(44/44);  
5     }  
6 }
```

On the right side of the IDE, a preview window shows the output "Powered 1".

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 IDE interface. At the top, there is a toolbar with a menu icon, a key icon, a Java logo with 'beta' text, a 'Run' button with a play icon, a share icon, and a dropdown arrow. Below the toolbar, a tab labeled 'Hello.java' with a gear icon is active. 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 }
```

On the right side of the IDE, a 'Power' window shows the number '1'.

Java Intro

Basics Lesson

Chapter 2 - Variable Types Integer & Modulo Division Strings

Chapter 2

- 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



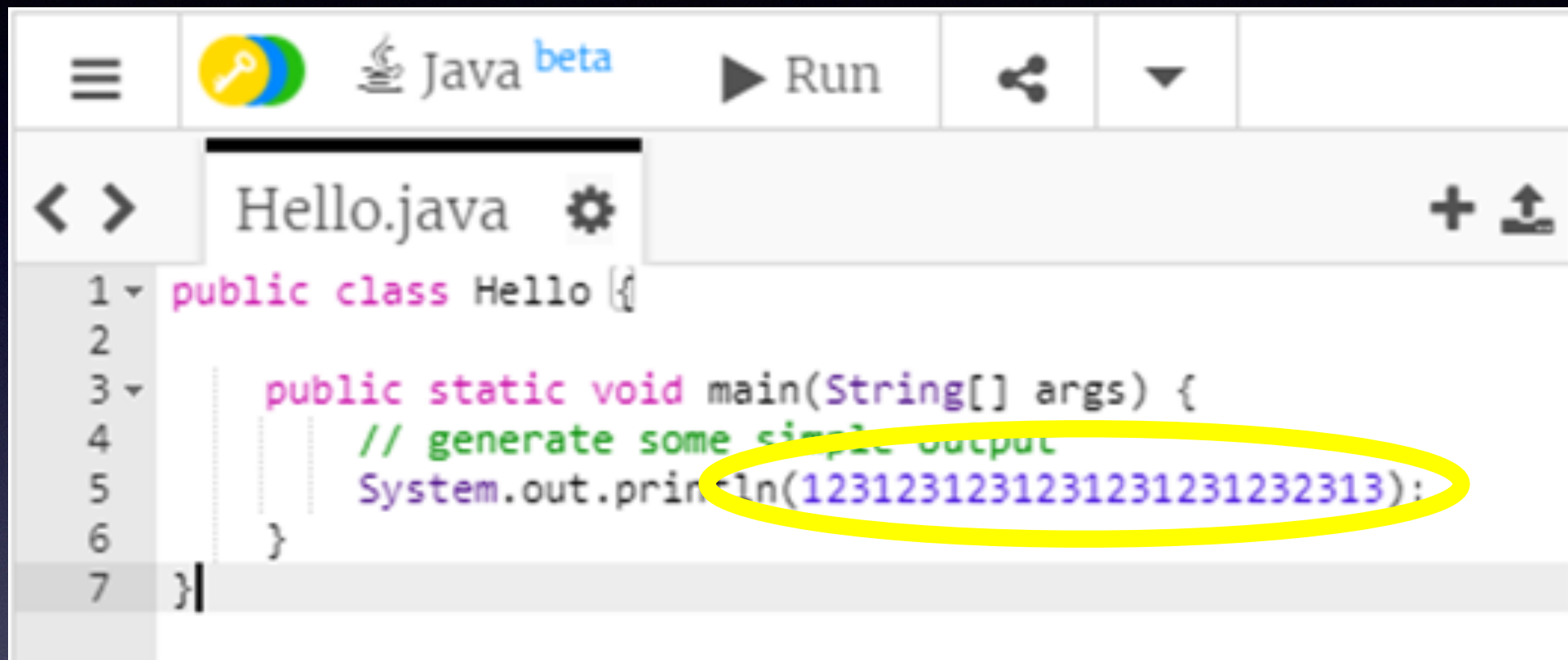
Variables: Parking space analogy

- 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



```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         // generate some simple output  
5         System.out.println(1231231231231231231232313);  
6     }  
7 }
```

Powered by  **trinket**

Hello.java:5: error: integer number too large.
1231231231231231231232313

System.out.println(1231231231231231231231232313),
^

1 error

Primitive data types

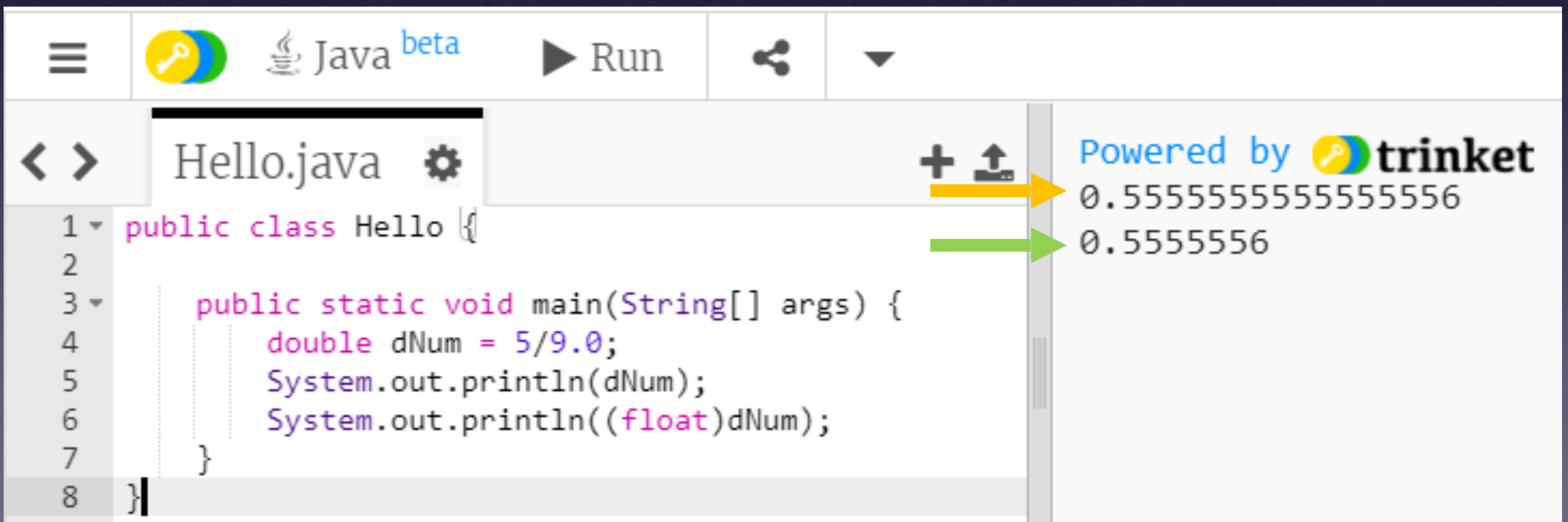
- In Java (unlike Python or JavaScript) variables have *types*
- Each type has size limits
- For this class, you need to know 5 basic (aka “primitive”) types:
 - `int`
 - `float`
 - `double`
 - `boolean`
 - `char`

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**



The screenshot shows a Java IDE with a file named `Hello.java`. The code defines a `public class Hello` with a `main` method. Inside the `main` method, a `double` variable `dNum` is assigned the value `5/9.0`. The program then prints `dNum` using `System.out.println(dNum);` and casts it to a `float` using `System.out.println((float)dNum);`. The output on the right shows the full precision of the `double` (0.5555555555555556) and the truncated precision of the `float` (0.5555556).

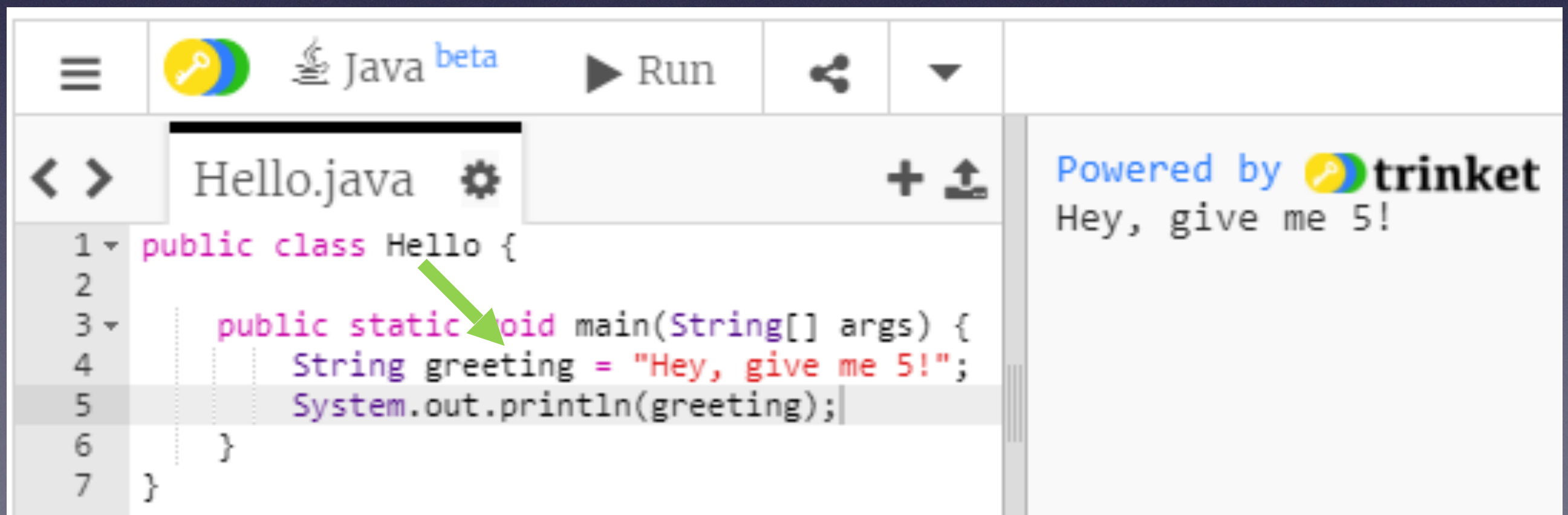
```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         double dNum = 5/9.0;  
5         System.out.println(dNum);  
6         System.out.println((float)dNum);  
7     }  
8 }
```

Powered by  **trinket**

0.5555555555555556
0.5555556

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 (")



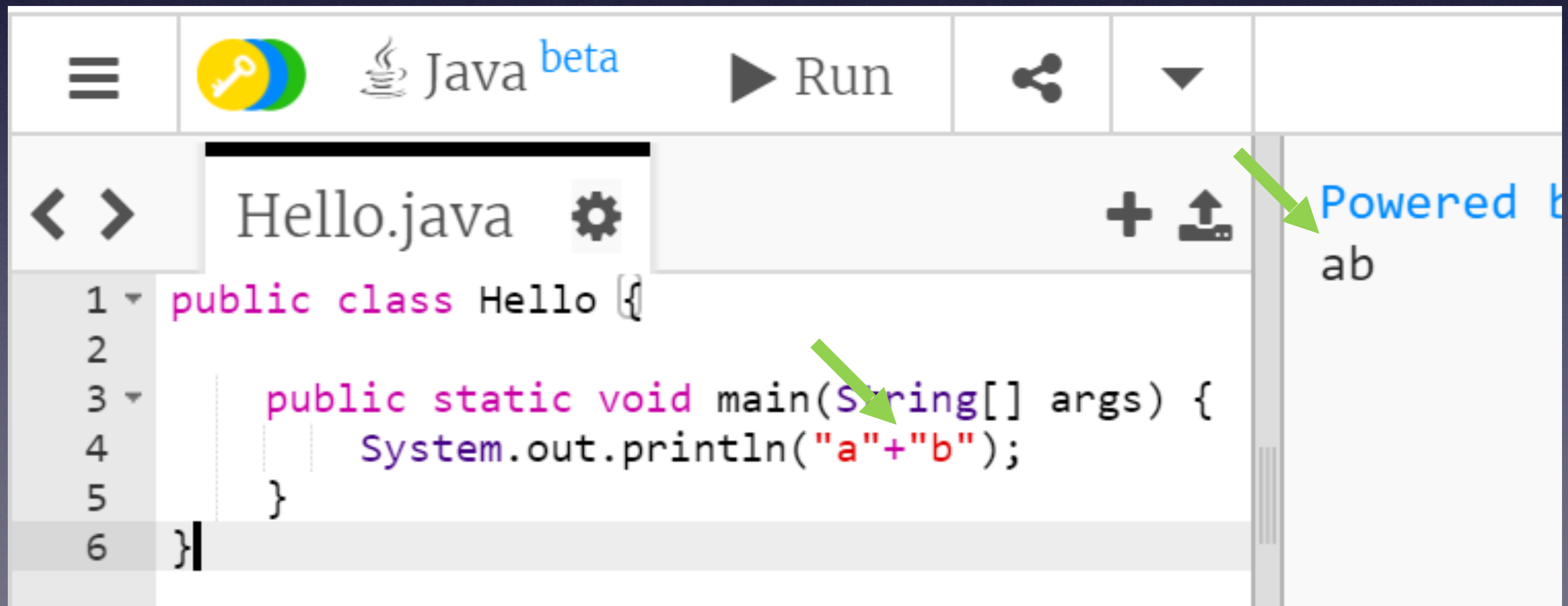
The screenshot shows a Java IDE interface. At the top, there's a toolbar with a menu icon, a key icon, a Java logo with 'beta' text, a 'Run' button, a share icon, and a dropdown arrow. Below the toolbar, the file 'Hello.java' is open, with a gear icon for settings and a '+ ↑' icon for file operations. The code in the editor is as follows:

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

A green arrow points from the text 'Hey, give me 5!' in the code to the output on the right. The output area on the right says 'Powered by trinket' and displays 'Hey, give me 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 a Java IDE interface. At the top, there's a toolbar with icons for a menu, a key, a coffee cup (Java logo), the text 'Java beta', a 'Run' button, a share icon, and a dropdown arrow. Below the toolbar, a tab labeled 'Hello.java' with a gear icon is active. The code editor shows the following Java code:

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

Two green arrows point from the code to the output. One arrow points from the `"a"+"b"` expression in the `println` statement to the output area. The other arrow points from the `+` operator in the same expression to the output area. The output area on the right shows the text 'Powered b' and 'ab'.

+ and Strings

- Java executes from left to right so, $1 + 2$ is 3, and $3 + \text{"Hello"}$ is "3Hello"
- $\text{"Hello"} + 1$ is "Hello1", and $\text{"Hello1"} + 2$ 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)

The screenshot shows a Java IDE with a file named 'Hello.java'. The code is as follows:

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

A red arrow points to the empty string "" in the code. On the right side of the IDE, the output is displayed as 'Power 11', with a green arrow pointing to the '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

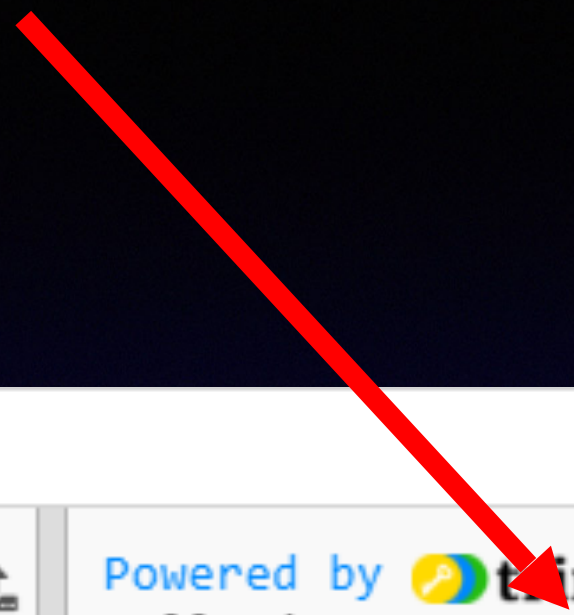
Variable names and keywords







- 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?



  Java beta  Run    Copy

  Hello.java   

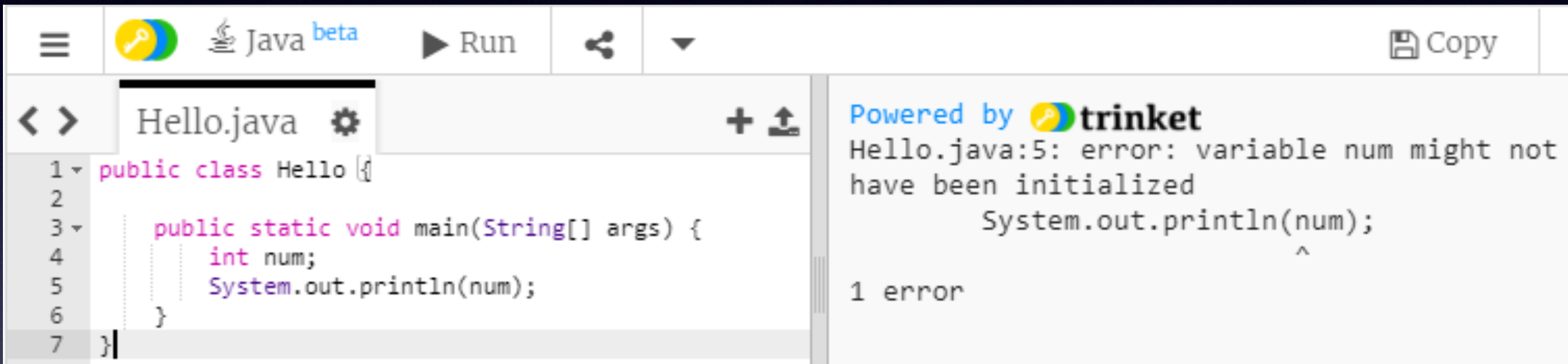
```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         int num;  
5         System.out.println(num);  
6     }  
7 }
```

Powered by  **trinket**

Hello.java:5: error: variable num might not
have been initialized
 System.out.println(num);
 ^
1 error

Variable initialization

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



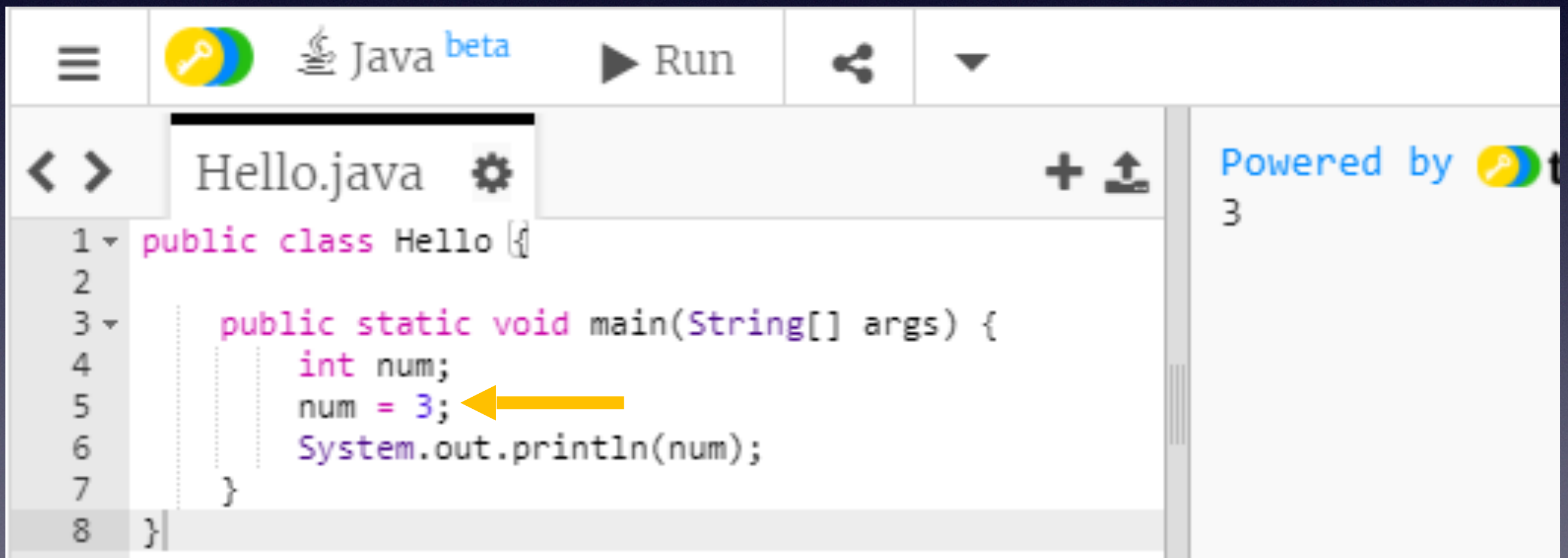
The screenshot shows a web-based Java IDE interface. At the top, there's a toolbar with icons for a menu, a key (representing code), a Java logo with 'beta' text, a 'Run' button, a share icon, and a dropdown arrow. On the right of the toolbar is a 'Copy' button. Below the toolbar, the editor window is titled 'Hello.java' with a settings gear icon and a '+ ↑' icon. The code in the editor is as follows:

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

On the right side of the editor, there's a console area. It says 'Powered by trinket' with the trinket logo. Below that, it displays an error message: 'Hello.java:5: error: variable num might not have been initialized'. The error points to the line 'System.out.println(num);' with a caret under the variable 'num'. At the bottom of the console, it says '1 error'.

Variable initialization

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



The screenshot shows an IDE window titled "Hello.java" with a "Run" button and a "Java beta" label. The code is as follows:

```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         int num;  
5         num = 3;  
6         System.out.println(num);  
7     }  
8 }
```

A yellow arrow points to the assignment statement `num = 3;` on line 5, highlighting the initialization of the variable `num`. The right sidebar shows "Powered by" and the number "3".

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?

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

Modulus and integer division

Remember how you did math in grade school?

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

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

$$\frac{8}{5}$$

$$8 \% 5$$

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

Kahoot!

Integer Division in Java