Computer Programming in Java
Bush School *CPJava Fall 2021*
Block D

Welcome to CPJava!

Chandru Narayan
Bicyclist Astronomer Engineer
Teaching Math 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 2021 SEMESTER	**12-Weeks**	**Sep to Dec 2021**
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 2022 SEMESTER	**15-Weeks**	**Jan to May 2022**
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 Exercises
Submitting Program Assignments
Projects
Professionalism & Integrity

AP CSA exam is on May 4th 2022
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 GR 204
  - 3:10 - 3:30 following CPJava Class
  - Extra Conf Hours by Calendly Appt via Portal
- Class Assistance
  - Ethan Mondri 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

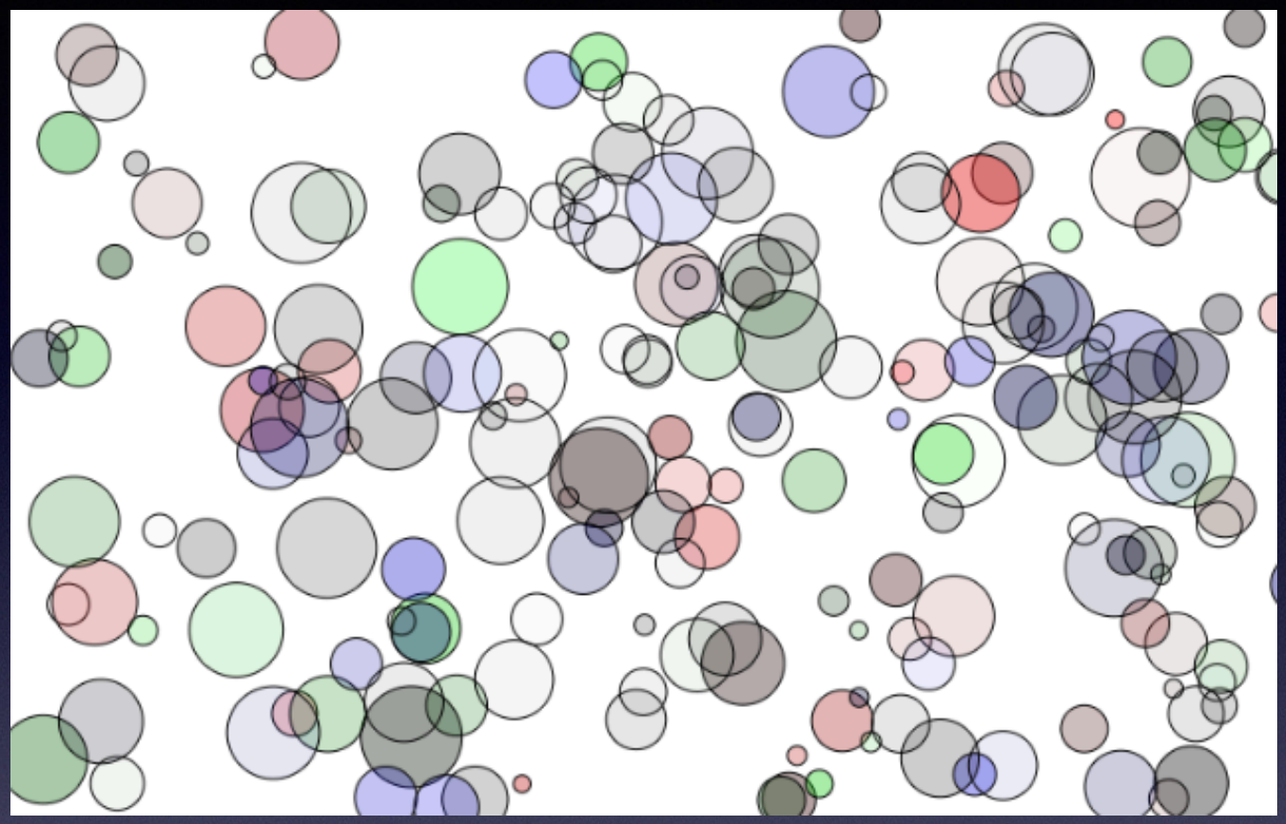
System Requirements

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!



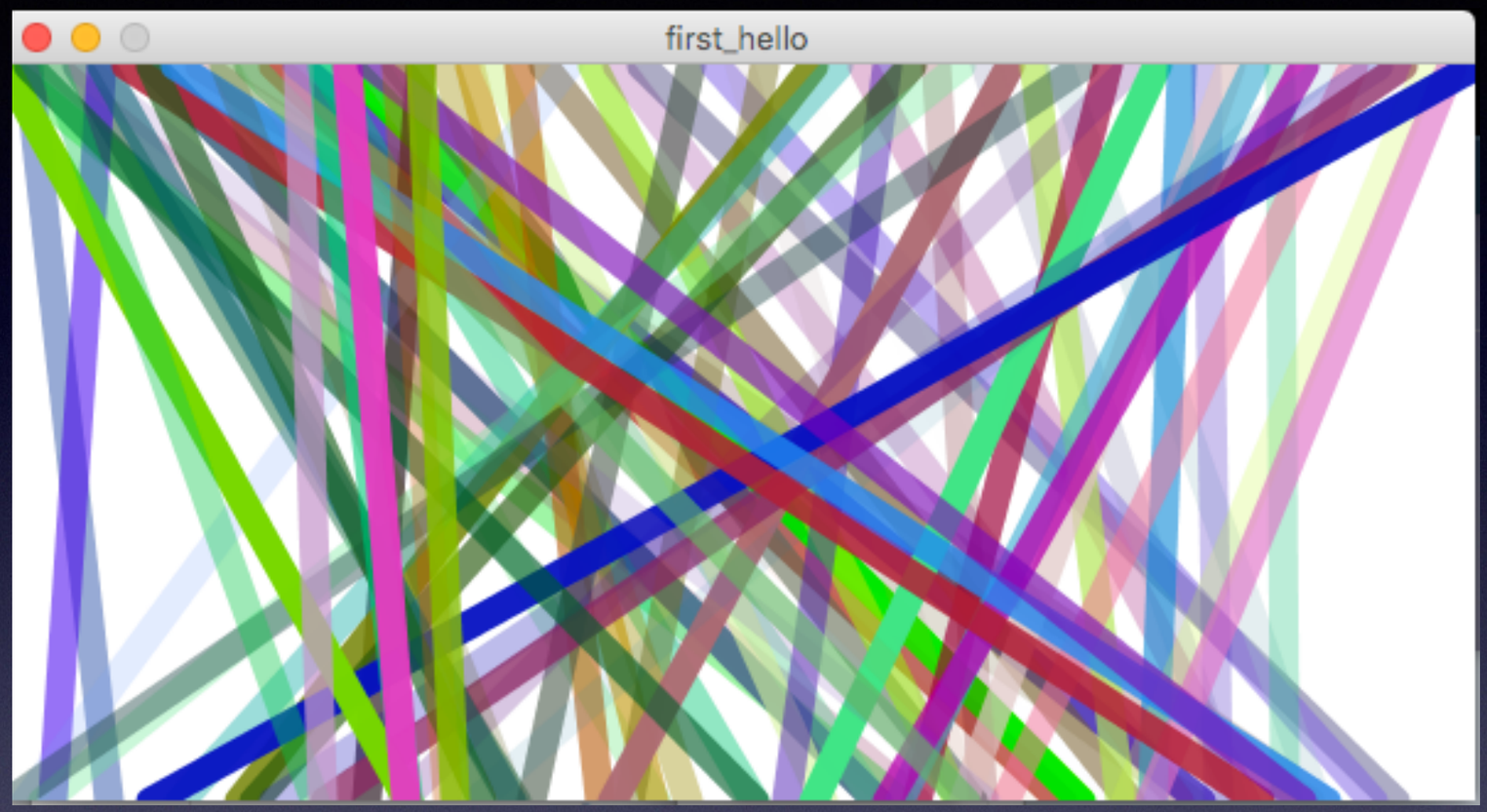
Bubbles - click to run the program

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

CSAwesome

Unit 1: 1.1-1.2

Unit 1: 1.1-1.2

The common building blocks in programming languages are:

Variables

Loops

if statements

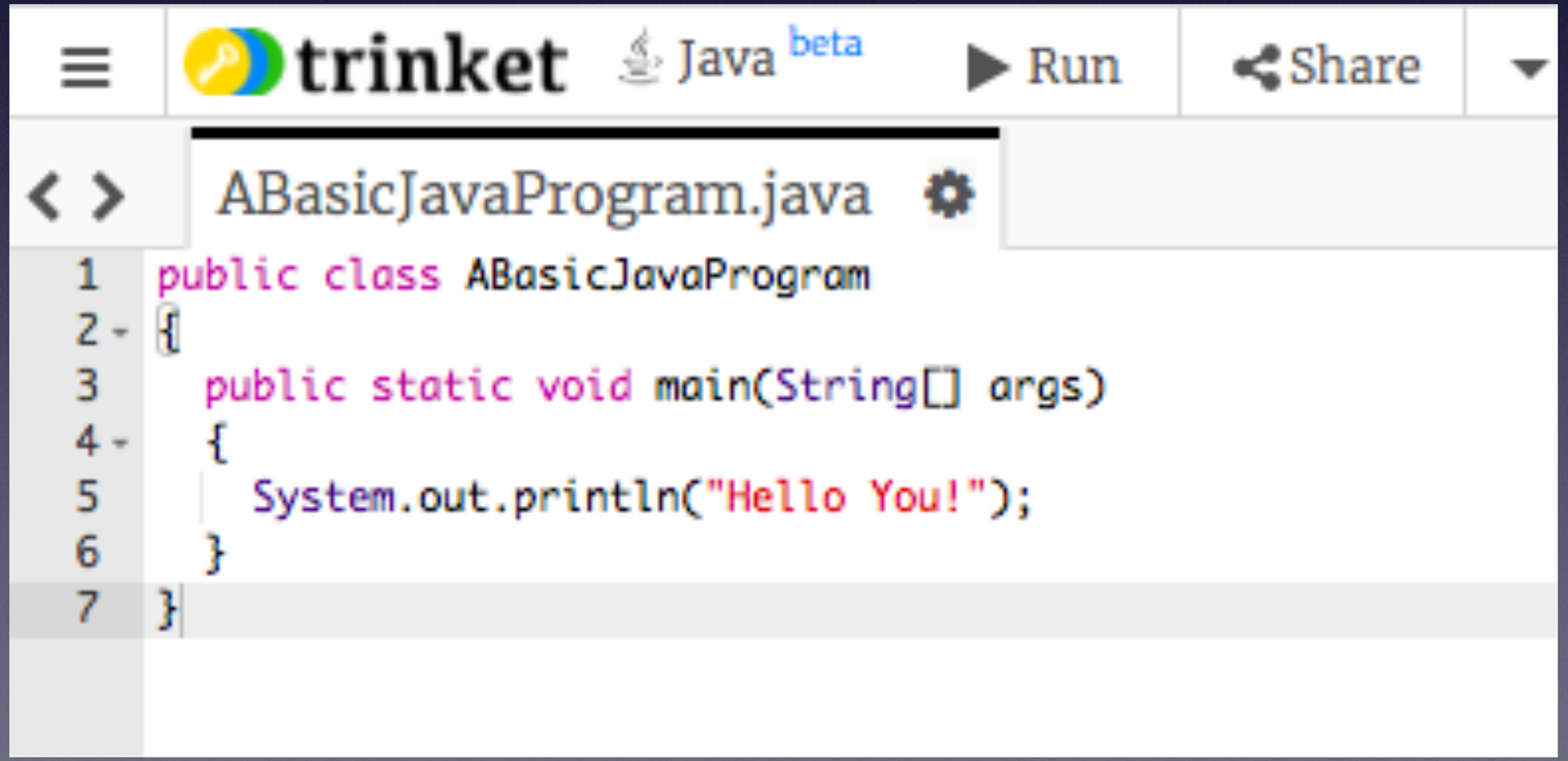
Functions (aka “methods”)

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 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 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 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
```

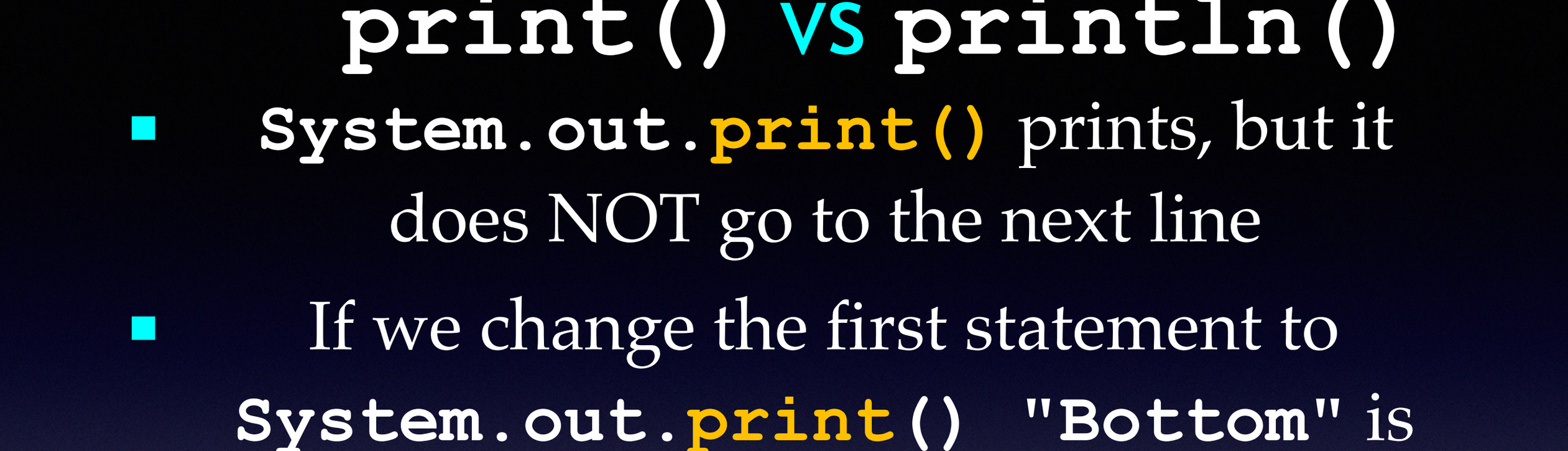
Result

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

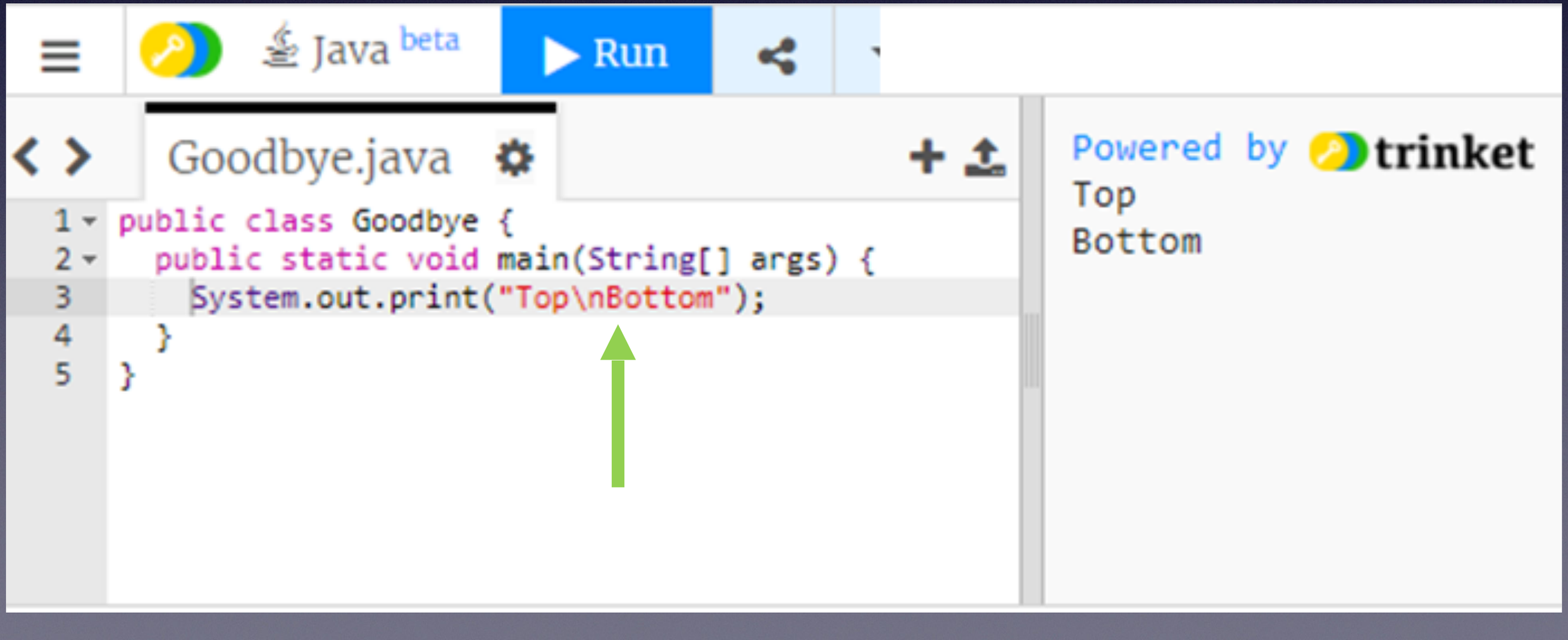
```
1  public class Hello {
2
3      public static void main(String[] args) {
4          // generate some simple output
5          System.out.println("Hello, World!");
6      }
7  }
```

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 (/)*

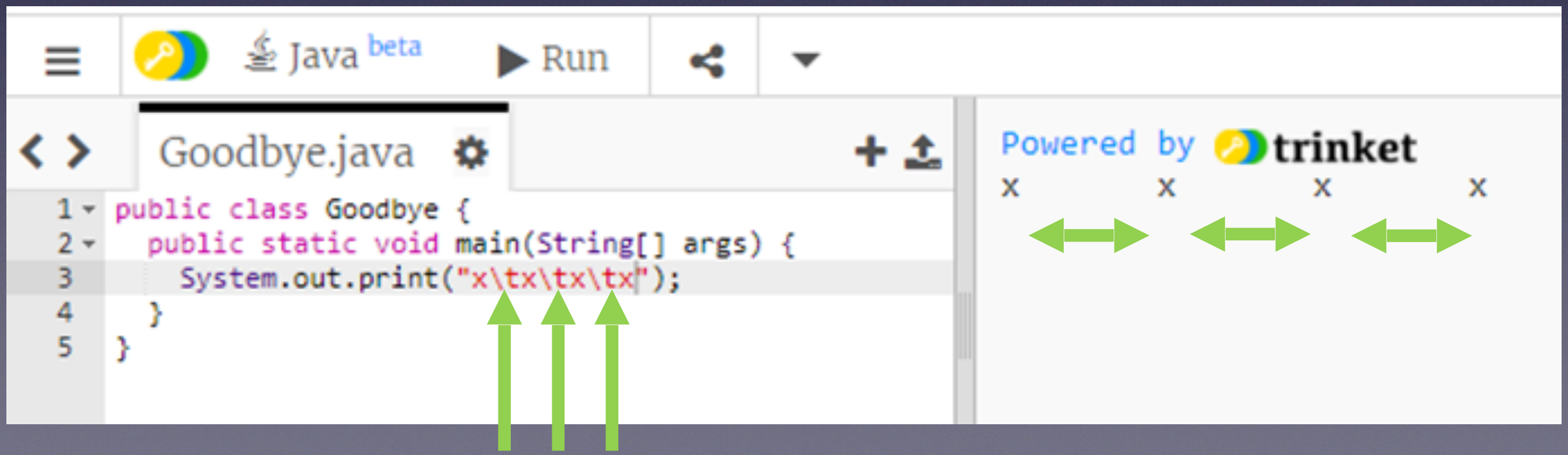
Common Java Escape Sequences

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



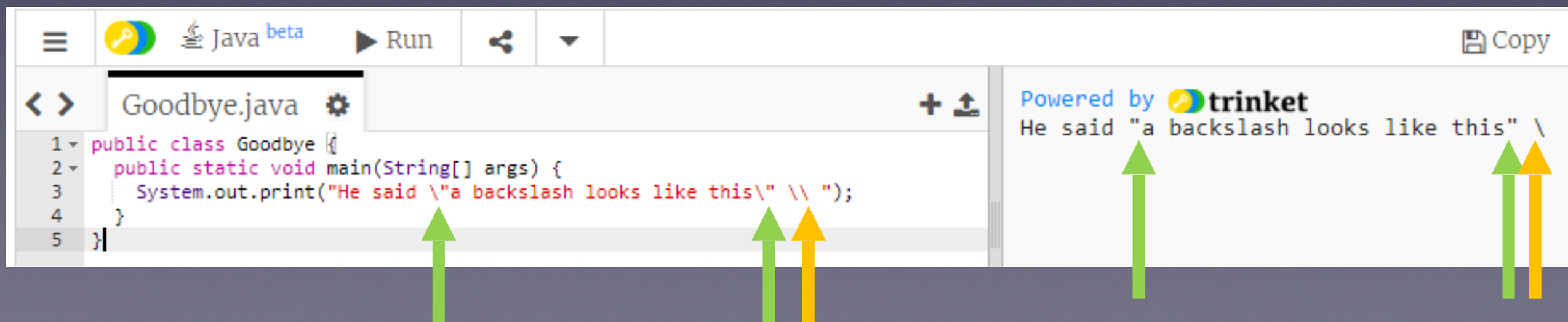
Common Java Escape Sequences

- `\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



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

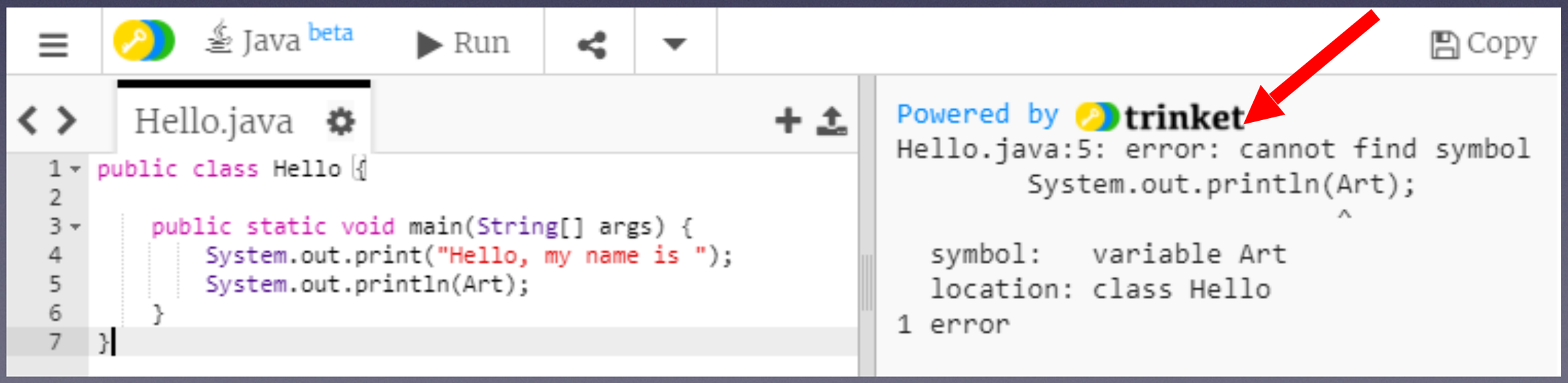
```
public class Goodbye {
    public static void main(String[] args) {
        System.out.print("Goodbye, ");
        System.out.println("cruel world");
    }
}
```

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

Java beta ▶ Run

Hello.java

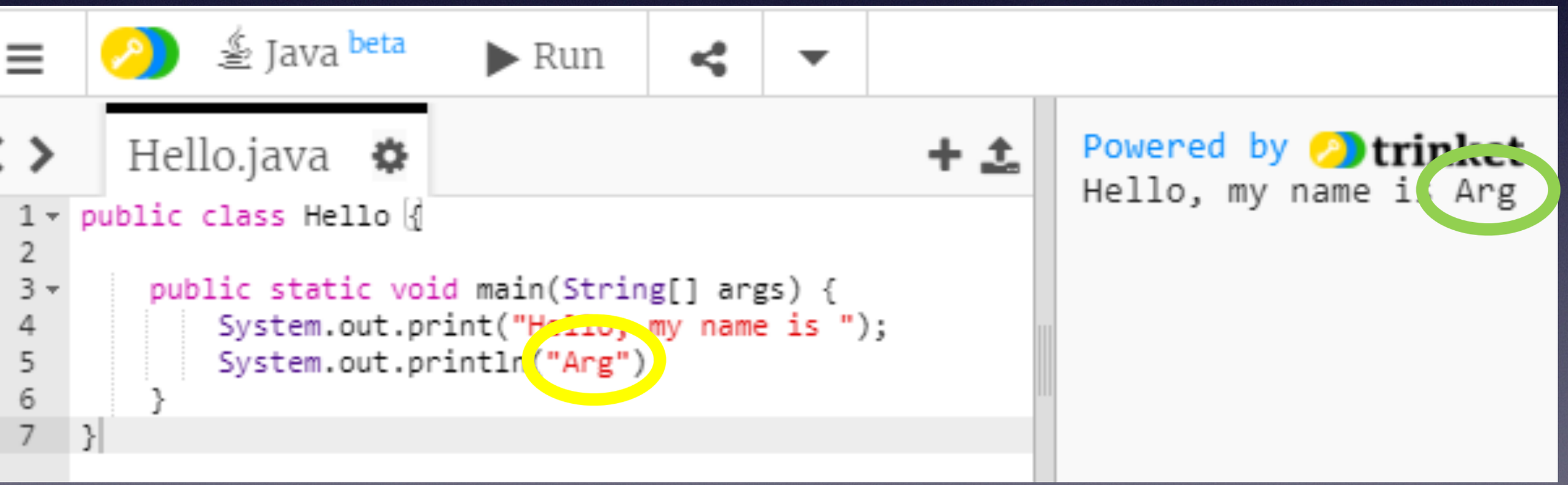
```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.print("Hello, my name is ");
5          System.out.println(Art);
6      }
7  }
```

Powered by trinket

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

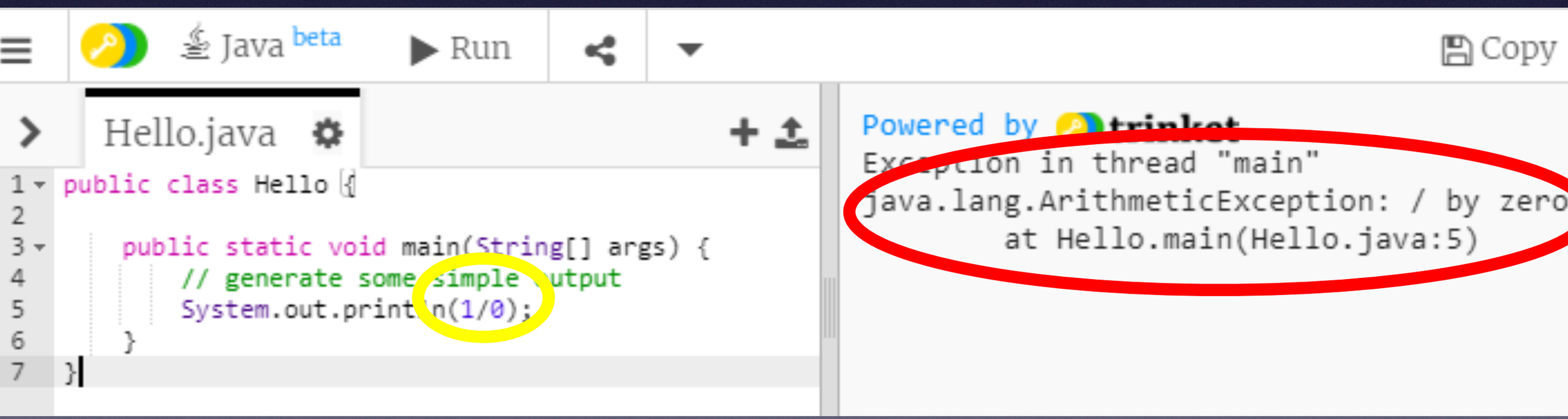
Logic error

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



(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

\+ – * /

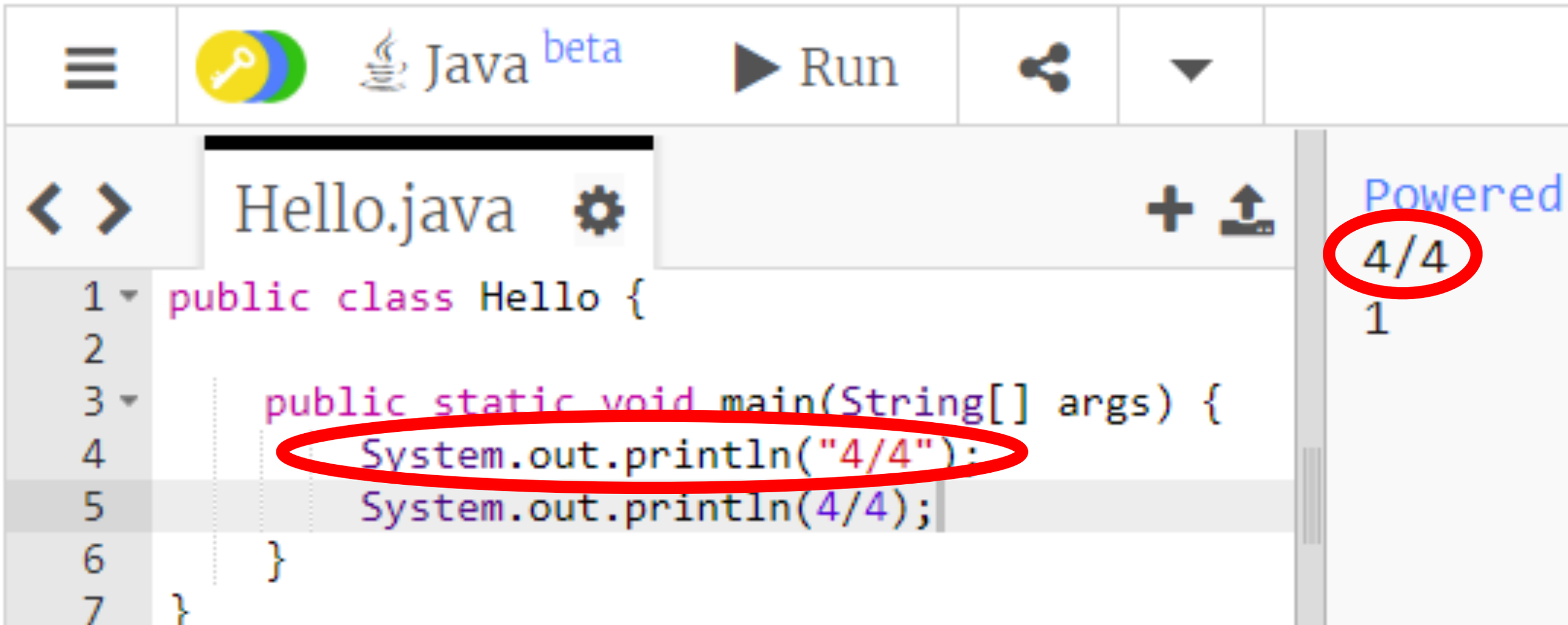
- Addition
- Subtraction
- Multiplication
- Division

Literals vs. Expressions

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

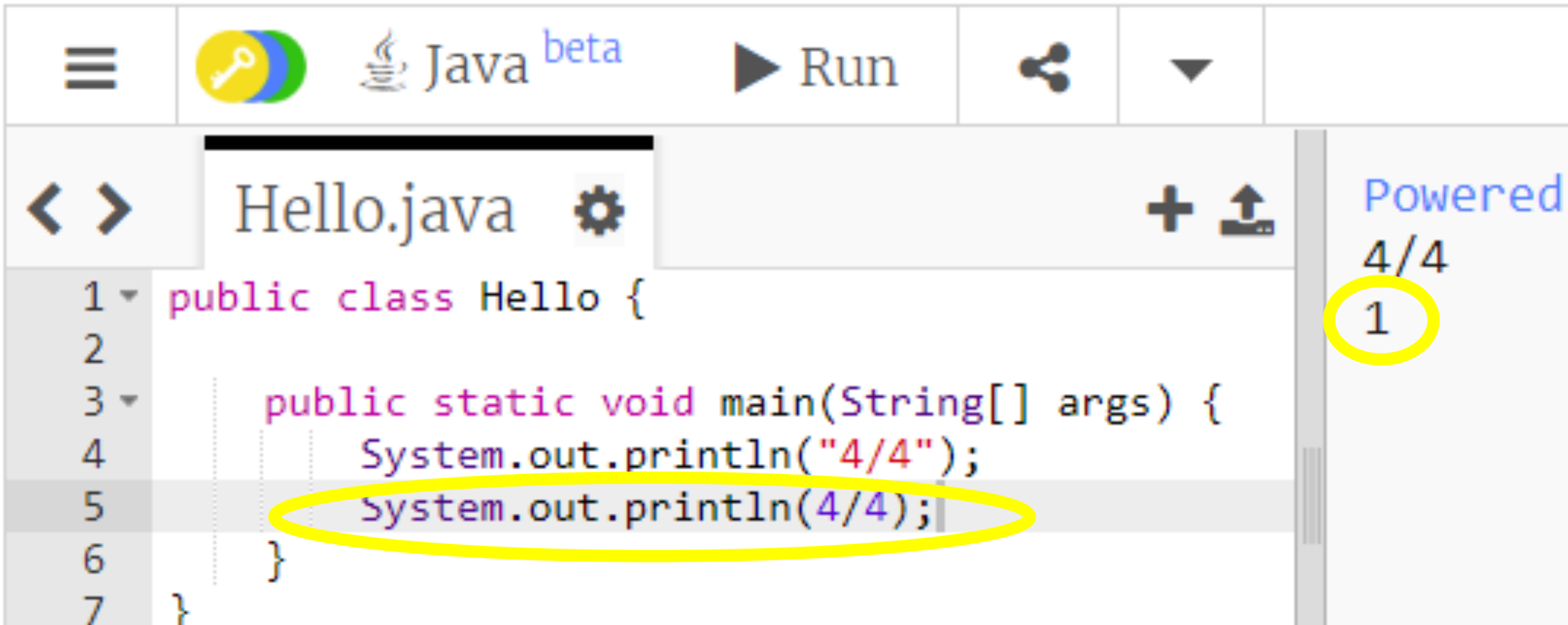
Literals vs. Expressions

- 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

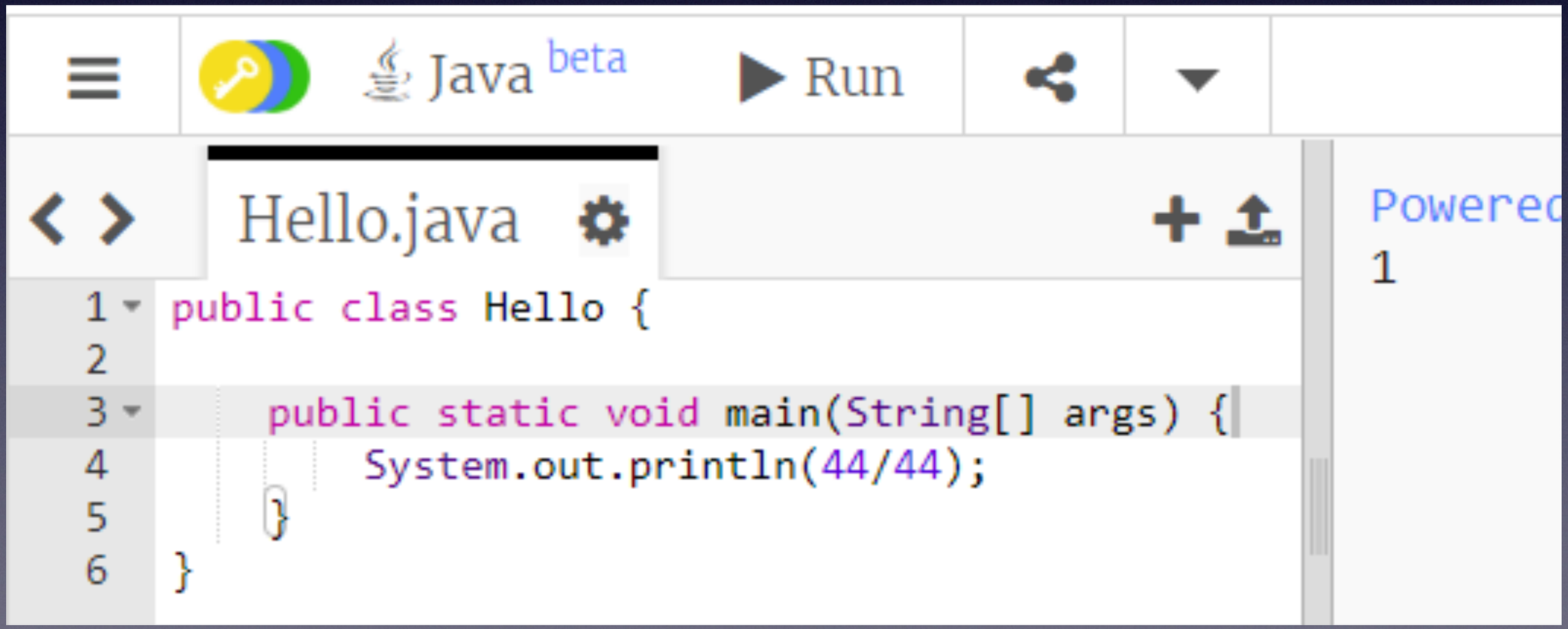


Four 4s challenge

- 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.

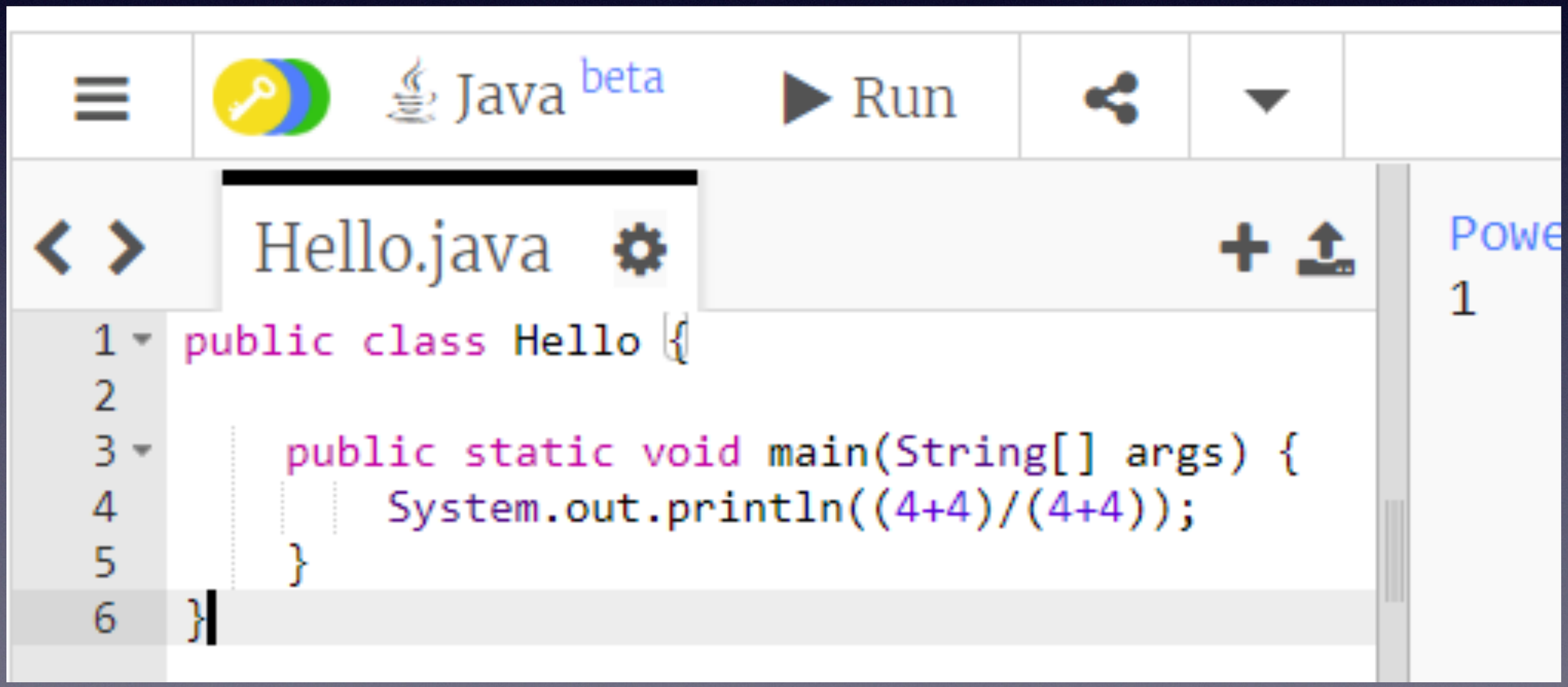
Four 4s challenge

- 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 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`**

Java beta ▶ Run

NewLine.java

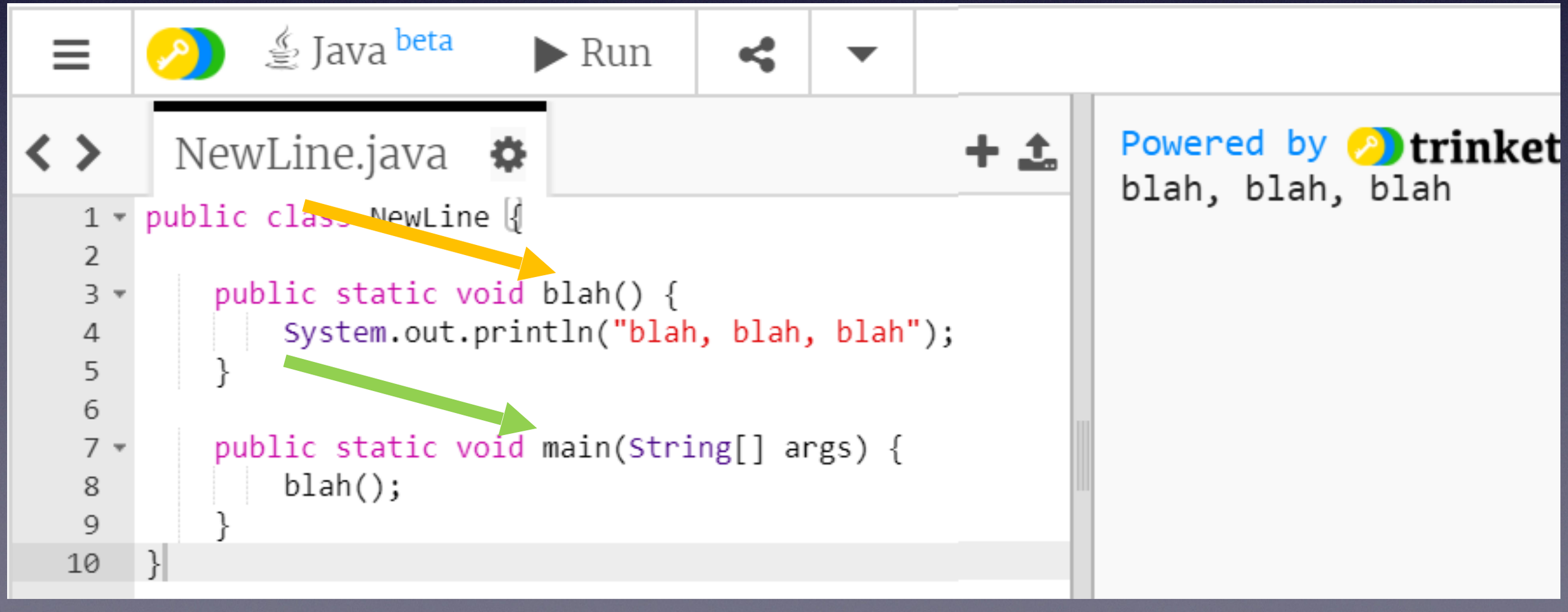
```
1  public class NewLine {
2
3      public static void blah() {
4          System.out.println("blah, blah, blah");
5      }
6
7      public static void main(String[] args) {
8          blah();
9      }
10 }
```

Powered by trinket

```
blah, blah, blah
```

`void` functions (aka methods)

- 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`**

Java beta ▶ Run

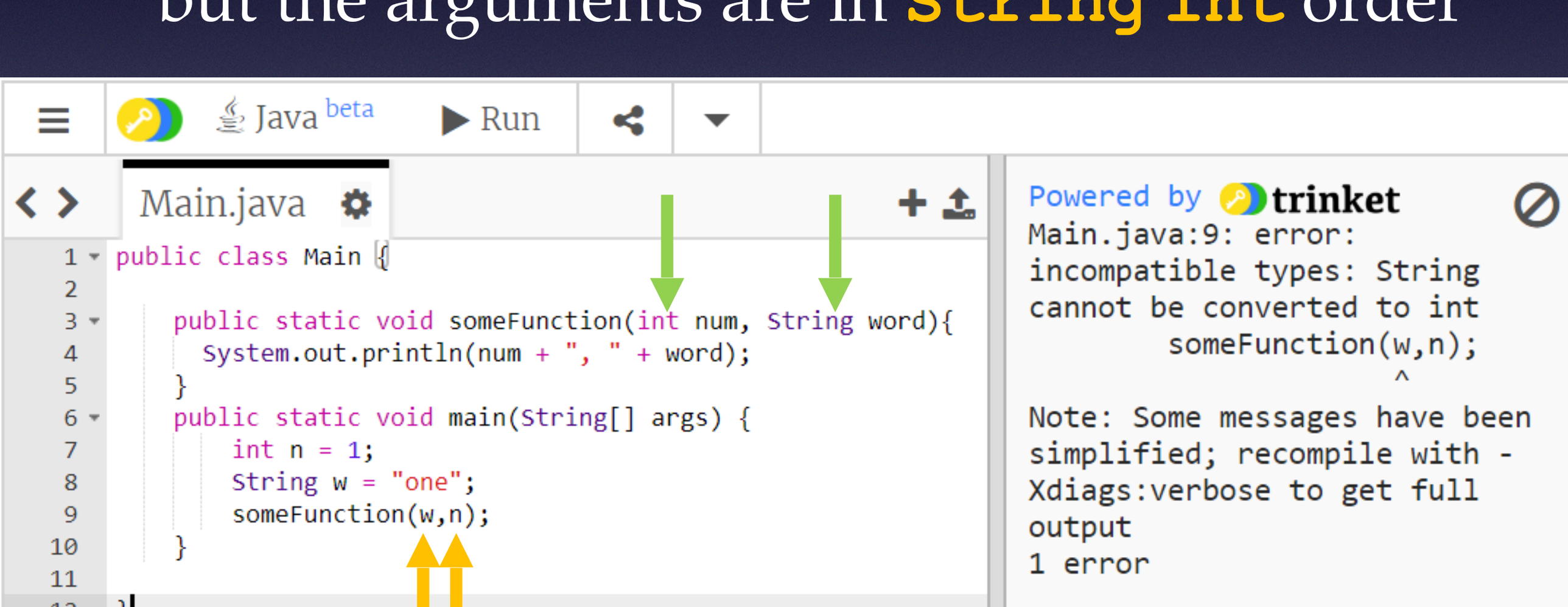
PrintTime.java

```
1  public class PrintTime {
2
3      public static void printTime(int hour, int minute) {
4          System.out.print(hour);
5          System.out.print(":");
6          System.out.println(minute);
7      }
8
9      public static void main(String[] args) {
10         int h = 7;
11         int m = 35;
12         printTime(h, m);
13     }
14 }
```

```
Powered b
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



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

Go to Unit 1 Exercises

Complete the PrintTime Function Assignment from the Functions and Parameters section

trinket Java beta Run Share

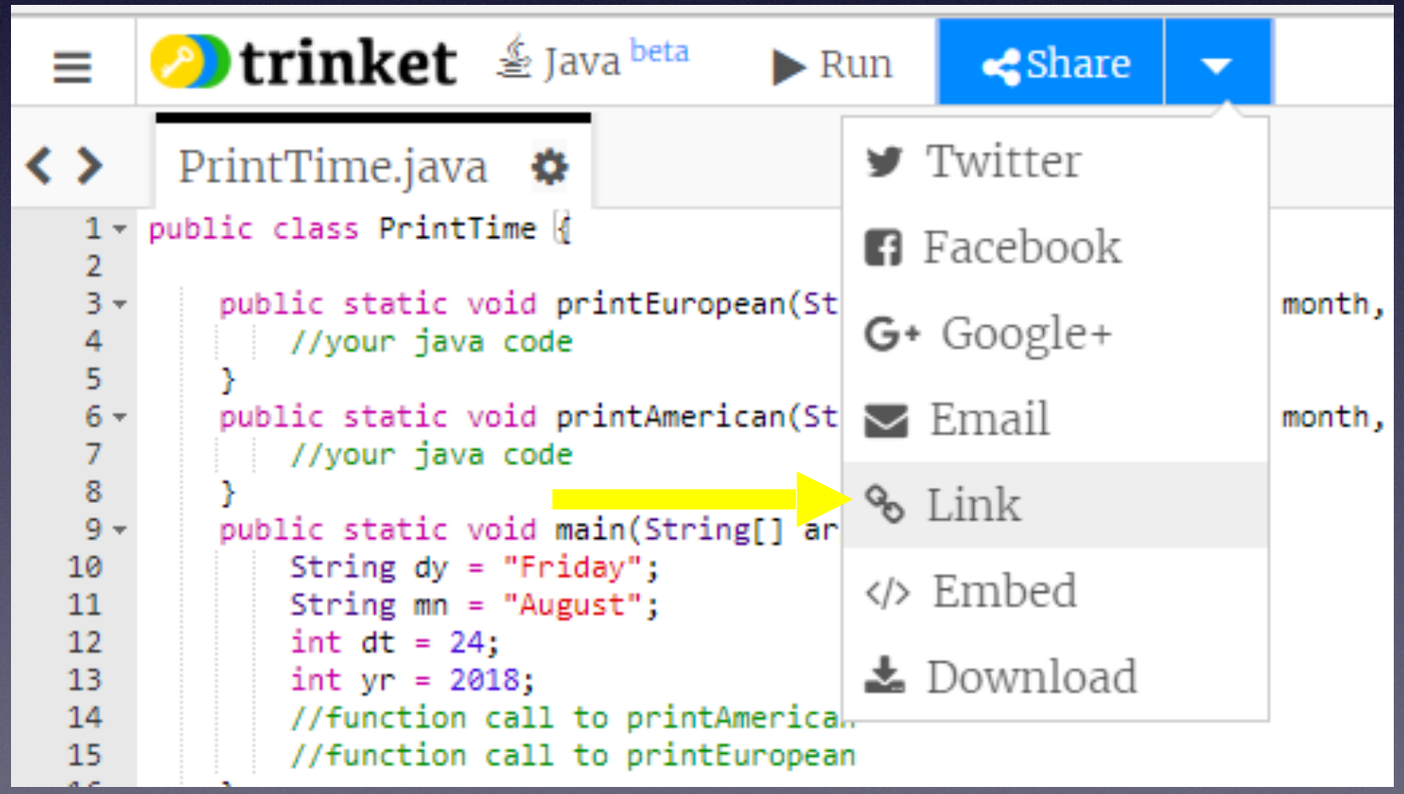
PrintTime.java

```
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



Kahoot!

End of first week AP Java

Variables and Expressions

CSAwesome

Unit 1: 1.3-1.4

Unit 1: 1.3-1.4

- 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

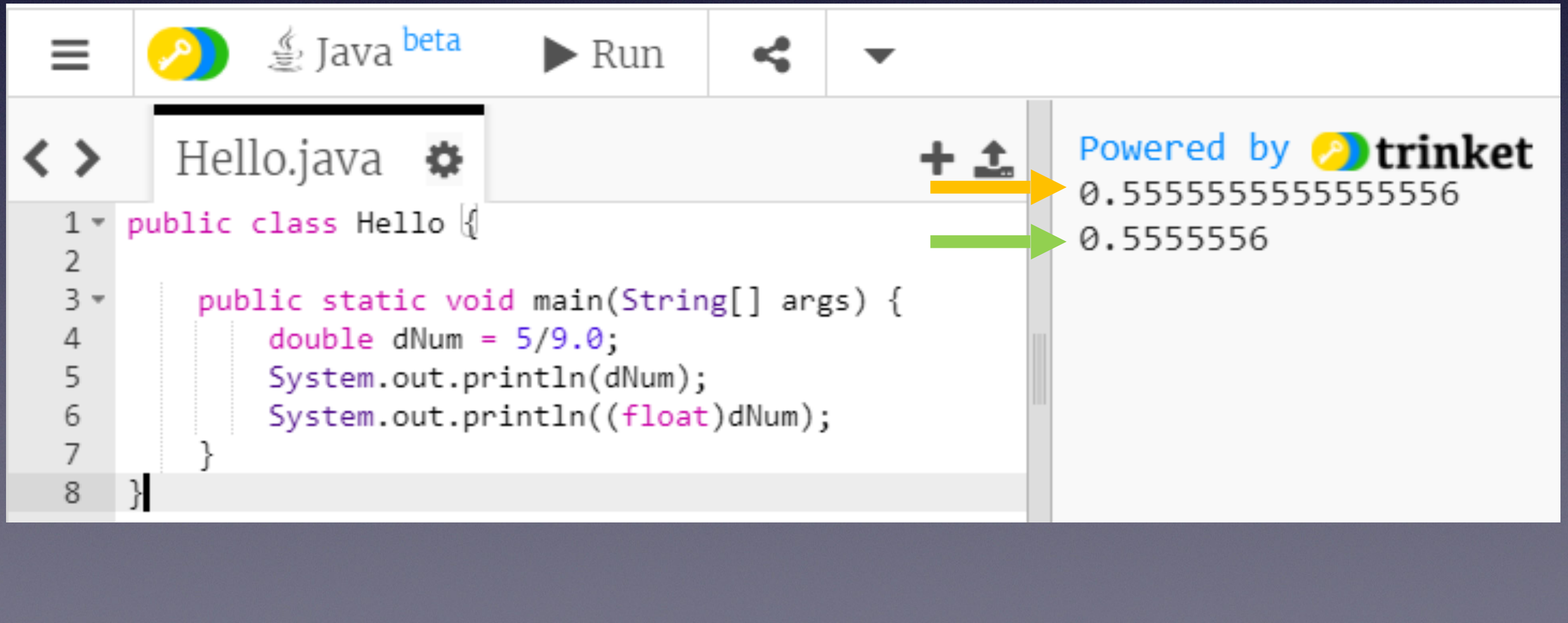
- In Java (unlike Python or JavaScript) variables have ***types***
- Each type has size limits
- For this class, you need to know 5 basic (aka "primitive") types:
  - **`int`**
  - **`float`**
  - **`double`**
  - **`boolean`**
  - **`char`**

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'`**

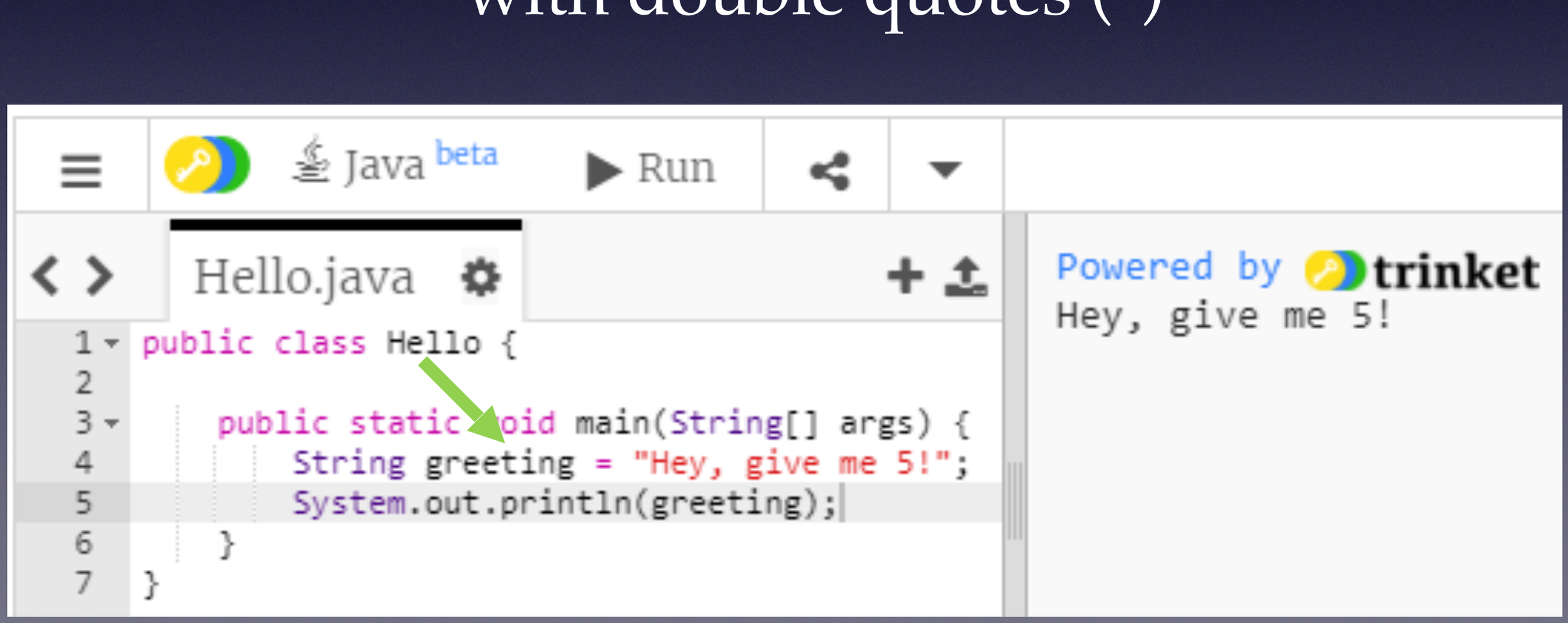
`float` vs. `double`

- `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`



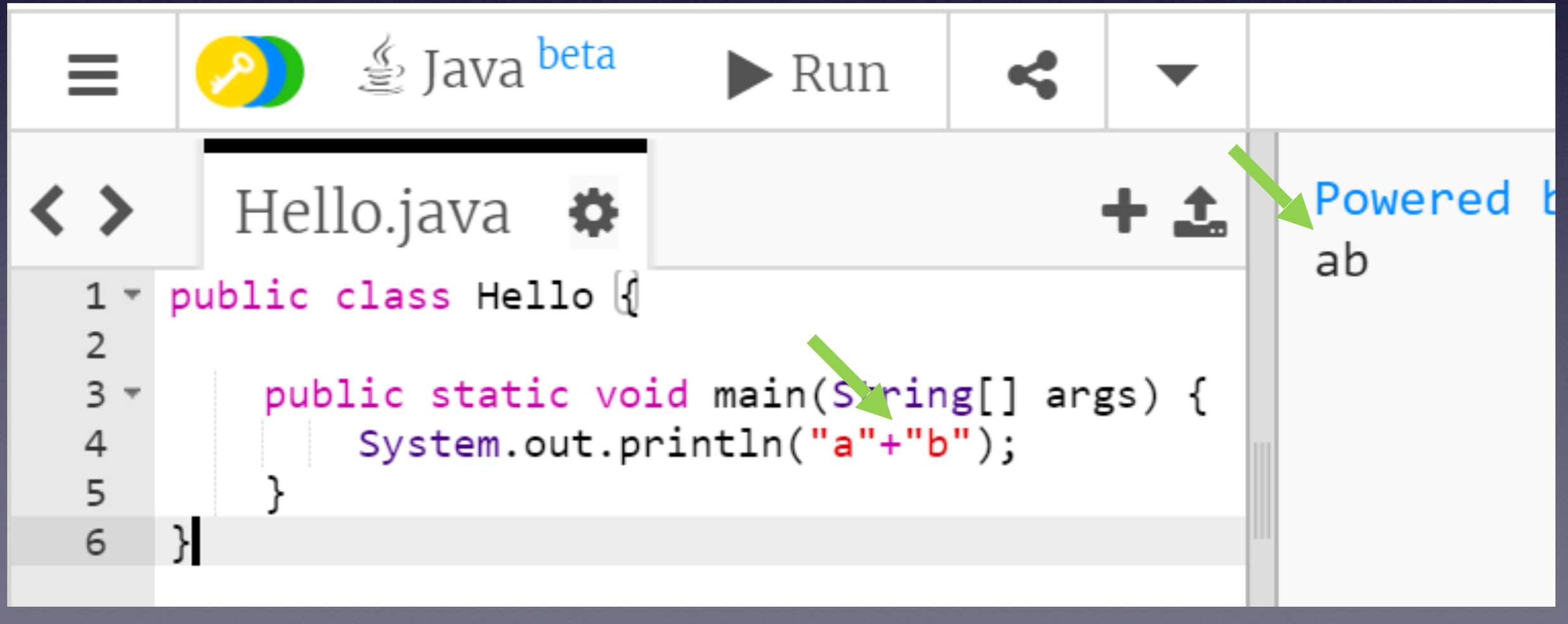
`String` variables

- 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

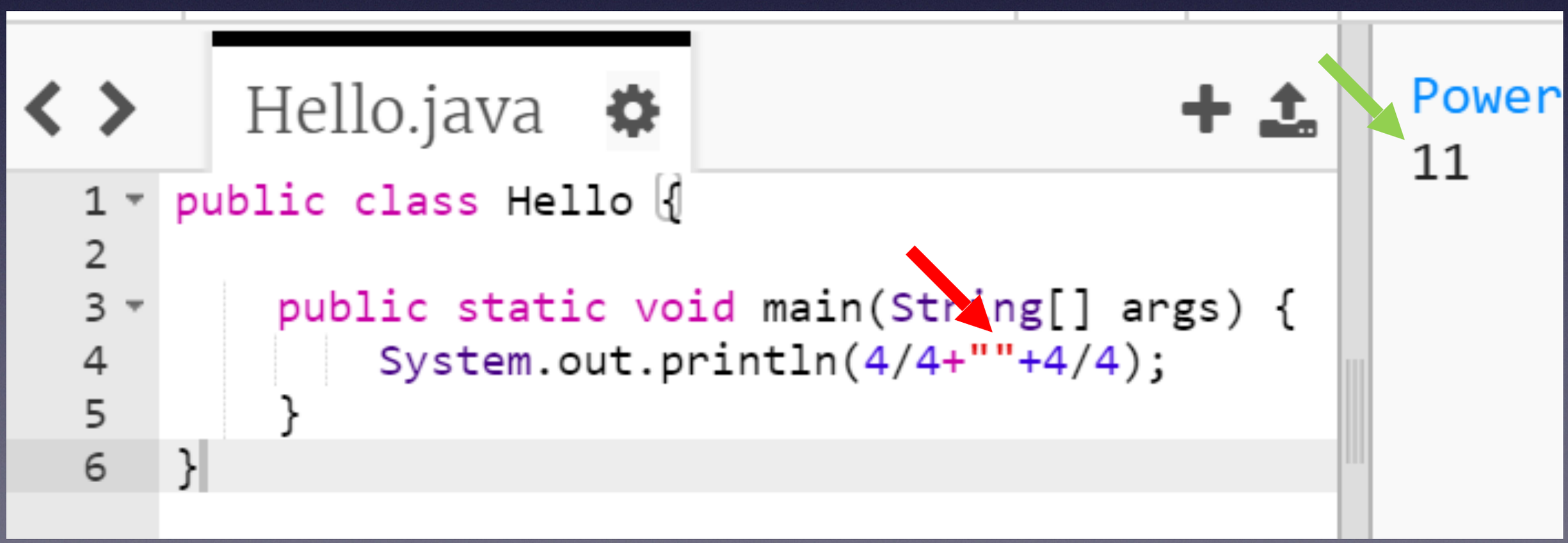
- Java executes from left to right so, `1 + 2` is `3`, and `3 + "Hello"` is `"3Hello"`
- `"Hello" + 1` is `"Hello1"`, and `"Hello1" + 2` is `"Hello12"`

```
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

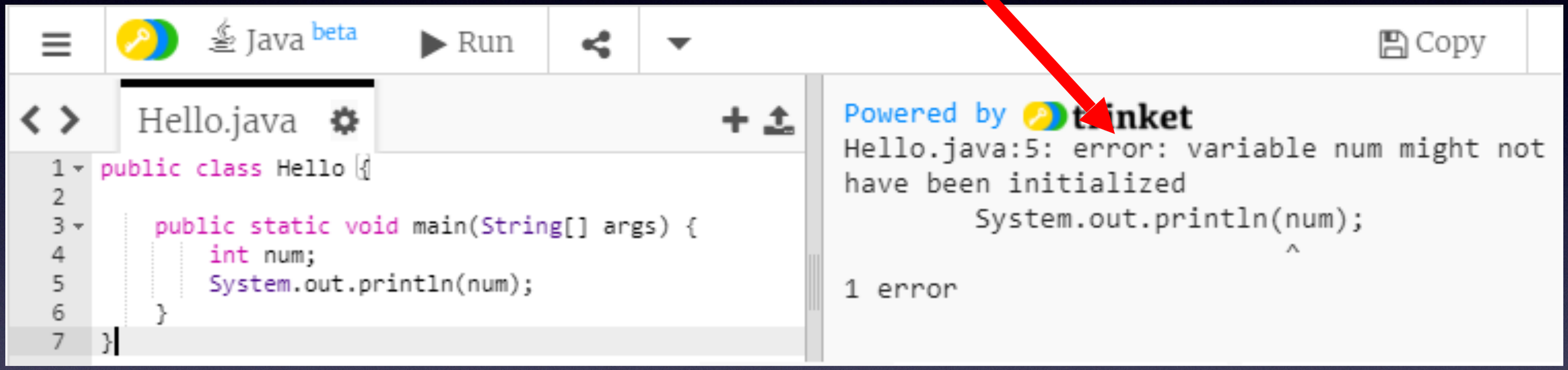
Variable names and keywords

- 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*

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

- 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

$$
\begin{array}{r}
0 \\
8\overline{)\,5} \\
0 \\
\hline
5
\end{array}
$$

In class: integers vs. decimals

- Java does division differently depending on the types of numbers
- When the types are mixed, the Java uses the larger type

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println(9/5);
5          System.out.println(9.0/5);
6          System.out.println(9/5.0);
7          System.out.println(9.0/5.0);
8      }
9  }
```

Power

```
1
1.8
1.8
1.8
```

Arithmetic Operators

- 3 **`int`** variables **`hour`**, **`minutes`**, **`second`**
- You may want to create more variables as well
- Do calculations to display:
  - Time in seconds since last midnight
  - Seconds remaining until midnight tonight
  - Percentage of day passed (hint: you will need to use at least one decimal number)

Hello.java

```
1  public class Hello {
2
3      public static void main(String[] args) {
4        int hour = 18, minute = 58, second = 10;
5
6        System.out.println("Time since midnight: " + ??);
7        System.out.println("Time remaining today: " + ??);
8        System.out.println("Percentage of day passed: " + ??);
9
```

Powered by trinket

```
Time since midnight: 68290 seconds
Time remaining today: 18110 seconds
Percentage of day passed: 0.7903935185185185%
```

Input with `Scanner`

- `import java.util.Scanner;`
  - Add scanner code to program
- `Scanner in = new Scanner(System.in);`
  - create a **`new Scanner`**
- `int someInteger = in.nextInt();`
  - Get an integer and store it in **`someInteger`**

Echo.java

```
1  import java.util.Scanner;
2
3  public class Echo {
4
5      public static void main(String[] args) {
6          Scanner in = new Scanner(System.in);
7
8          System.out.print("Type an integer: ");
9          int someInteger = in.nextInt();
10         System.out.println("You typed: " + someInteger);
11
12     }
```

```
Powered by trinket
Type an integer:  42
You typed: 42
```

The types must match

- `String someInteger = in.nextInt();`
- Java won't let you store an **`int`** in a String

Echo.java

```
1  import java.util.Scanner;
2
3  public class Echo {
4
5      public static void main(String[] args) {
6          Scanner in = new Scanner(System.in);
7
8          System.out.print("Type an integer: ");
9          String someInteger = in.nextInt();
10         System.out.println("You typed: " + someInteger);
11
12     }
13 }
```

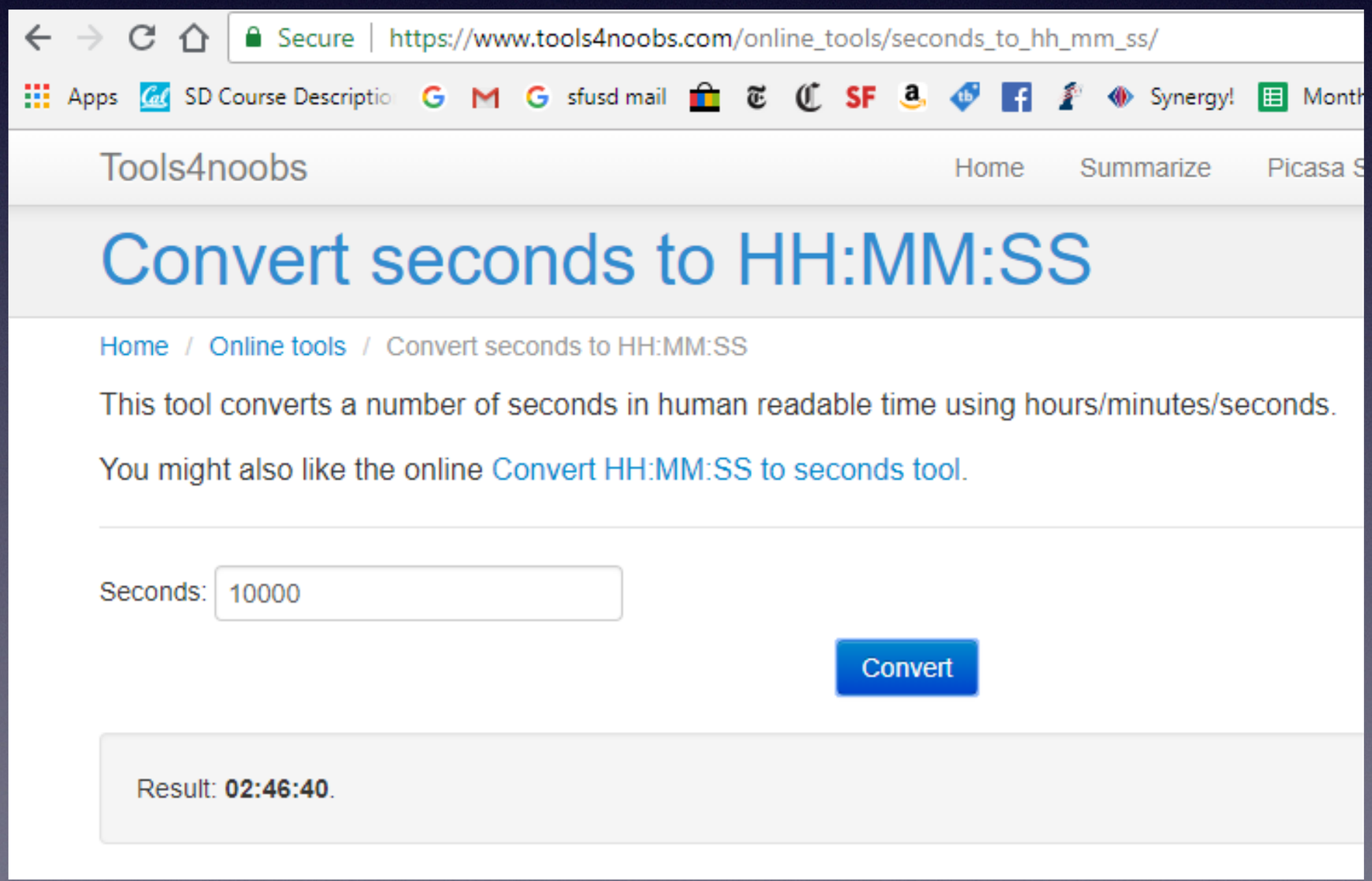
Powered by trinket

```
Echo.java:9: error:
incompatible types: int
cannot be converted to
String
          String someInteger
= in.nextInt();

^
1 error
```

An online time converter

- www.tools4noobs.com/online_tools/seconds_to_hh_mm_ss/
  - 10,000 seconds is 2 hours 46 minutes and 10 seconds



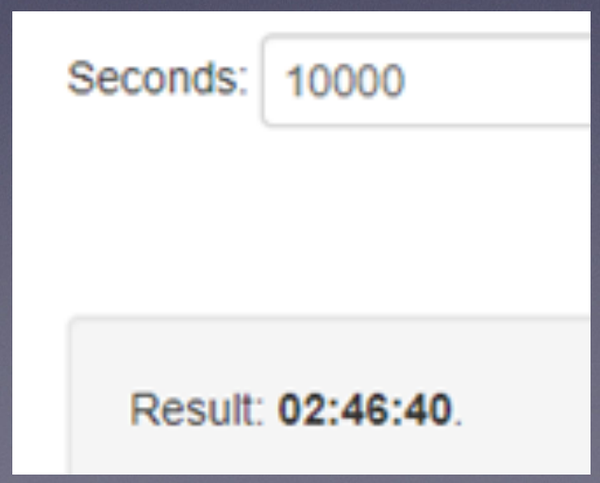
Using modulus

10,000 / (60*60) is 2 (hours)

10,000 % (60*60) is 2,800

2,800 / 60 is 46 (minutes)

2,800 % 60 is 40 (seconds)



Time Converter Exercise

Write your own seconds to hours / minutes / seconds converter

You can use the Echo program in chapter 3 as a starting point

Echo.java

```
1  import java.util.Scanner;
2
3  public class Echo {
4
5      public static void main(String[] args) {
6          int seconds;
7          Scanner in = new Scanner(System.in);
8
9          System.out.print("Type a number of seconds: ");
10         seconds = in.nextInt();
11         System.out.println(seconds + " seconds is:");
12         int hours = seconds/(60 * 60);
13         seconds = seconds%(60*60);
14         System.out.println(hours + " hours");
15         //Your java code here
16     }
17 }
```

Powered by trinket

```
Type a number of seconds:
10000
10000 seconds is:
2 hours
46 minutes
and 40 seconds
```

Chapter 5

- `if` statements
- Conditions (expressions that evaluate to `true` or `false`)
- Logical operators
- Relational operators
- Short circuit evaluation
- Distributing the `!` (De Morgan's laws)
- We'll skip recursion and stack diagrams for now

Relational Operators

> >= < <= != ==

- Is Greater Than
- Is Greater Than or Equal
- Is Less Than
- Is Less Than or Equal
- Is Not equal
- Is Equal to

Logical operators

&& || !

- And
- Or
- Not
- Logical operators are sometimes also called *Boolean* operators

Truth Tables

- Truth tables show the results combining expressions with `&&` `||` or `!`

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

Logical Operators

- What would be printed?

```
int nCount = 150;
System.out.println((0<nCount)&&(nCount<=100));
```

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

Logical Operators

- What would be printed?

```
int nCount = 150;
System.out.println((0<nCount)&&(nCount<=100));
```

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

Hello.java

```
1
2  public class Hello {
3    public static void main(String[] args) {
4      int nCount = 150;
5      System.out.println(( 0 < nCount ) && ( nCount <= 100 ));
6    }
7  }
```

Powered by trinket

```
false
```

Logical Operators

- Now what would be printed?

```
int nCount = 50;
System.out.println((0<nCount)&&(nCount<=100));
```

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

Logical Operators

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

- What would be printed?

```
int nCount = 50;
System.out.println((0<nCount)&&(nCount<=100));
```

Hello.java

```
1
2  public class Hello {
3    public static void main(String[] args) {
4      int nCount = 50;
5      System.out.println(( 0 < nCount ) && ( nCount <= 100 ));
6    }
7  }
```

Powered by trinket

```
true
```

How would Java evaluate these expressions?

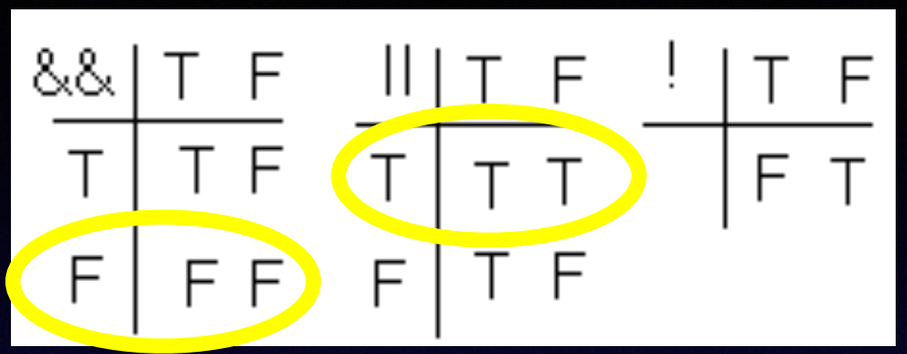
&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

```
int nCount = 150;
( 0 < nCount ) || ( nCount <= 100 )
                //true
char cGrade = 'B';
('A' <= cGrade) && (cGrade <= 'F')
                //true
int nAge = 16;
! (19 >= nAge && nAge >= 13)
                //false
int nAge = 16;
19 < nAge || nAge < 13
                //false
```

Short circuit evaluation



- **`true || `*`anything`*** is always **`true`**, so Java does not need to evaluate the expression ***`anything`***
- Likewise, **`false &&`** ***`anything`*** is always false
- Ignoring the ***`anything`***, when possible, is called **short circuit evaluation**

Short circuit evaluation

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

- What does
- `-3 < 0 || 567*2-17/3==1128`
- evaluate to?
- **`true`**
- Its not necessary to evaluate the second condition because the first condition is **`true`** and **`true || anything`** is always **`true`**

Short circuit evaluation

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	T	F
	F	T

- What does

```
1==0 && 274/17 > 7846/461
```

evaluate to?

- **`false`**
- Its not necessary to evaluate the second condition because the first condition is **`false`** and **`false && `*`anything`*** is always **`false`**

Distributing the !

- The rules for distributing the `!` are known as De Morgan's laws

`!(A && B)` *is the same as* `!A || !B`

`!(A || B)` *is the same as* `!A && !B`

Distributing the !

- More examples of De Morgan's laws

```
!(x < 5 && y == 3)
```

is the same as

```
x >= 5 || y != 3
```

```
!(x >= 1 || y != 7)
```

is the same as

```
x < 1 && y == 7
```

`if` statements

- If the condition in parentheses is **`true`**, the code in the curly braces (called a block) executes

```
if (x > 0) {
    System.out.println("x is positive");
}
```

`if else` statements

- If the condition in parentheses is **`true`**, the code in the first block executes otherwise the second block executes

```
if (x > 0) {
    System.out.println("x is positive");
}
else {
    System.out.println("x is NOT positive");
}
```

if else if statements

- The block under the first **true** condition runs, all other blocks are skipped
- If all conditions are **false**, the **else** block runs

```
if (temp > 80) {
System.out.println("Hot");
} else if (temp > 65) {
System.out.println("Warm");
} else if (temp > 50) {
System.out.println("Cool");
} else {
System.out.println("Cold");
}
```

if statements

- Curly braces are optional if a block has only one statement

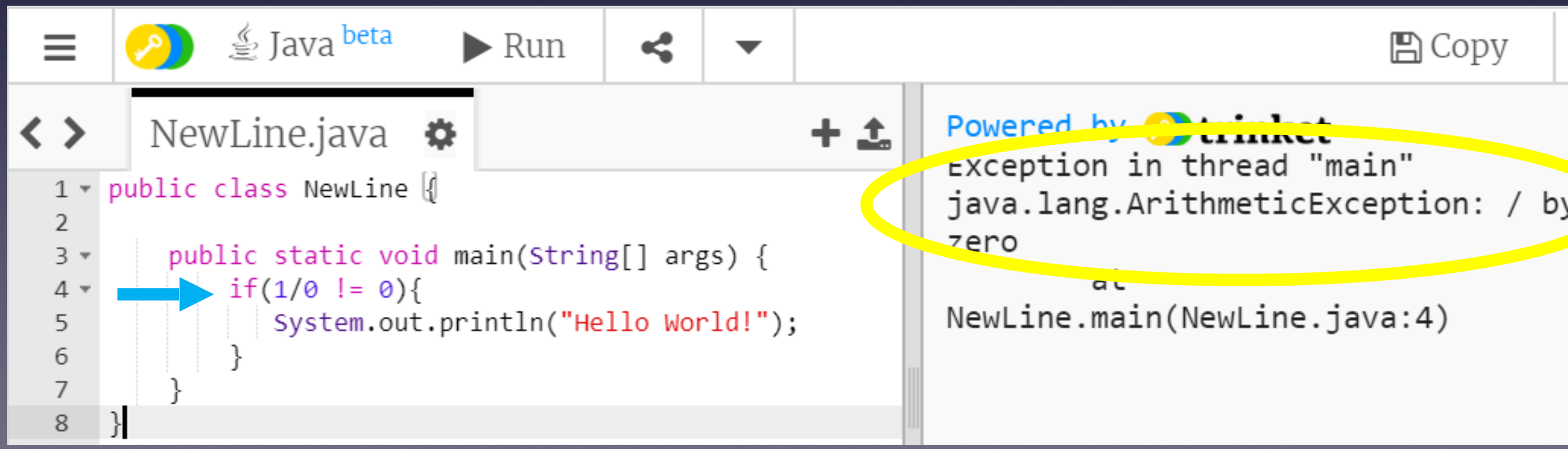
```
if (x > 0) {
    System.out.println("x is positive");
}
```

is the same as

```
if (x > 0)
    System.out.println("x is positive");
```

More short circuit evaluation

- Notice that **`Hello World!`** is not printed because of the divide by zero exception in the **`if(1/0 != 0)`**

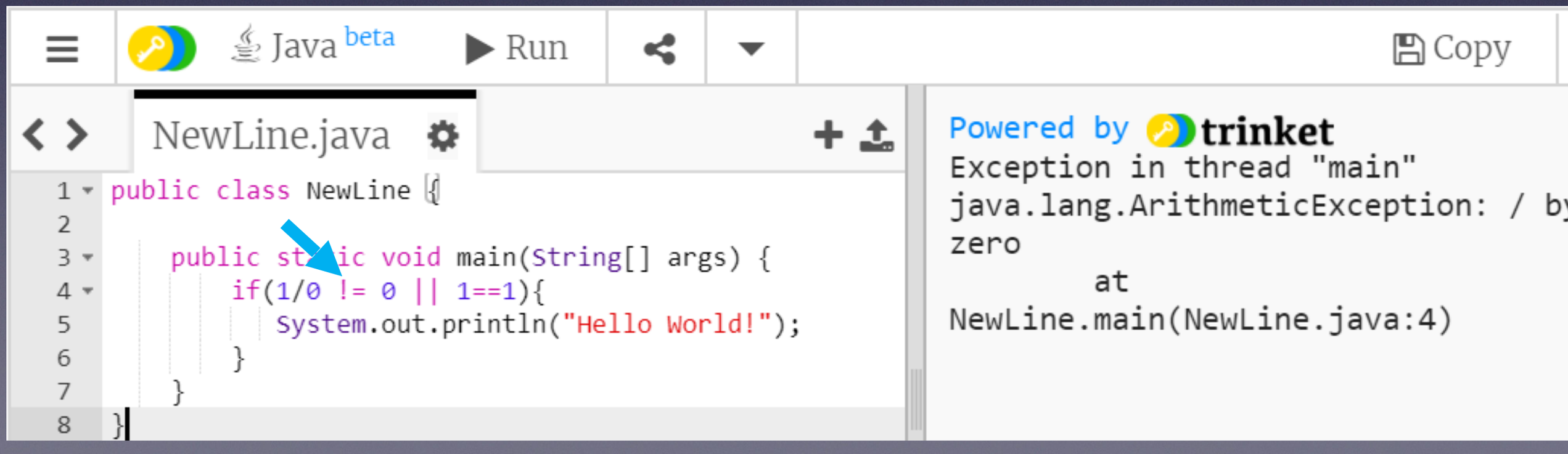


More short circuit evaluation

- Now **`Hello World!`** ***is*** printed and there is no divide by zero exception
- Because **`1==1 || anything`** is **`true`**, Java does not execute the code that would cause an exception

More short circuit evaluation

- In this version, the code that causes the exception executes first so **`Hello World!`** is not printed



Compound Boolean Exercise

- Combine the two `if` statements into one using a logical operator
- You can use a program like the one below to check your answer

NewLine.java

```
1  public class NewLine {
2
3    public static void main(String[] args) {
4      int x = 10;
5      if (x > 0) {
6        if (x < 10) {
7          System.out.println("positive single digit number.");
8        }
9      }
10   }
11 }
```

Matching names & curly braces

- For your program to work, these two names must match
- Your program must have a `main` function
- You need matching curly braces

NewLine.java

```
1  public class NewLine {
2
3    public static void main(String[] args) {
4      int x = 8;
5      if(x > 0 && x < 10){
6          System.out.println("positive single digit");
7      }
8    }
9  }
```

Powered by trinket

```
positive single digit
```

Functions & `return` values

- **`void`** functions are used for their *effect*
- Functions with another type other than **`void`** make a value of that type and **`return`** it
- We'll skip the other stuff in Chapter 6 (recursion, JavaDoc tags and overloading) for now

An `absoluteValue()` function

- Note **`double`** and **`return`** statements

```
public static double absoluteValue(double x) {
    if (x < 0) {
        return -x;
    } else {
        return x;
    }
}
```

A program using `absoluteValue()`

- Two function definitions
  `absoluteVaue()` and **`main()`**

Java beta ▶ Run

NewLine.java

```
1  public class NewLine {
2
3      public static double absoluteValue(double x) {
4          if (x < 0) {
5              return -x;
6          } else {
7              return x;
8          }
9      }
10     public static void main(String[] args) {
11       System.out.println(absoluteValue(-99) );
12       System.out.println(absoluteValue(0) );
13       System.out.println(absoluteValue(37) );
14     }
15 }
```

Powered by

```
99.0
0.0
37.0
```

A program using `absoluteValue()`

- Three function calls to `absoluteVaue()`

Java beta ▶ Run

NewLine.java

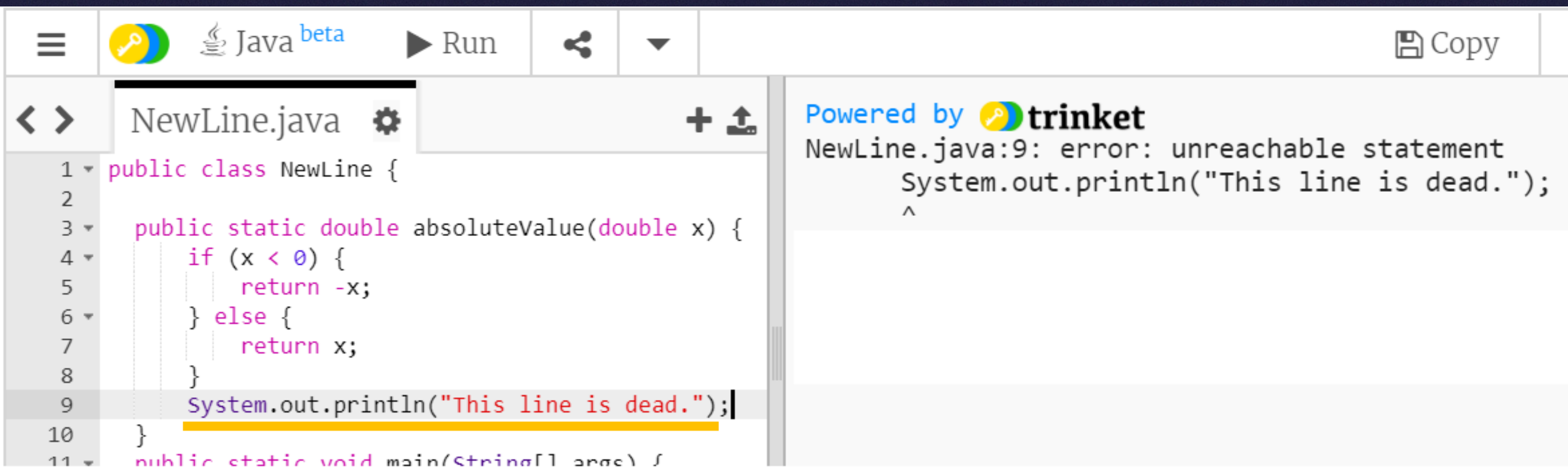
```
1  public class NewLine {
2
3    public static double absoluteValue(double x) {
4        if (x < 0) {
5            return -x;
6        } else {
7            return x;
8        }
9    }
10   public static void main(String[] args) {
11     System.out.println(absoluteValue(-99) );
12     System.out.println(absoluteValue(0) );
13     System.out.println(absoluteValue(37) );
14   }
15 }
```

Powered by

```
99.0
0.0
37.0
```

Unreachable "dead" code

- The **`System.out.println()`** is unreachable because it is *after* **`return`**
- **`return`** "goes back" and no other code in **`absoluteValue()`** is executed



An `isSingleDigit()` function

- A **`boolean`** function that **`returns`** **`true`** or **`false`**

Java beta ▶ Run

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isSingleDigit(int x) {
4      if (x > -10 && x < 10) {
5        return true;
6      } else {
7        return false;
8      }
9    }
10   public static void main(String[] args) {
11     System.out.println(isSingleDigit(-99) );
12     System.out.println(isSingleDigit(0) );
13     System.out.println(isSingleDigit(37) );
14     System.out.println(isSingleDigit(7) );
15   }
16 }
```

```
Powered
false
true
false
true
```

A program using `isSingleDigt()`

- Four function calls to **`isSingleDigit()`**

Java beta ▶ Run

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isSingleDigit(int x) {
4      if (x > -10 && x < 10) {
5          return true;
6      } else {
7          return false;
8      }
9    }
10   public static void main(String[] args) {
11     System.out.println(isSingleDigit(-99) );
12     System.out.println(isSingleDigit(0) );
13     System.out.println(isSingleDigit(37) );
14     System.out.println(isSingleDigit(7) );
15   }
16 }
```

```
Powered
false
true
false
true
```

A different `isSingleDigt()`

- Not better, just another way of doing the same thing

Java beta ▶ Run

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isSingleDigit(int x) {
4      return x > -10 && x < 10;
5    }
6
7    public static void main(String[] args) {
8      boolean result = isSingleDigit(-99);
9      System.out.println(result);
10     result = isSingleDigit(0);
11     System.out.println(result);
12   }
13 }
```

Powered

```
false
true
```

Storing the `return` value

- Again, just another way of doing the same thing

Java beta ▶ Run

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isSingleDigit(int x) {
4      return x > -10 && x < 10;
5    }
6
7    public static void main(String[] args) {
8      boolean result = isSingleDigit(-99);
9      System.out.println(result);
10     result = isSingleDigit(0);
11     System.out.println(result);
12   }
13 }
```

Powered

```
false
true
```

Review: Syntax error

- *Syntax* is the grammar and spelling of a computer language
- Here I forgot the double quotes around my name which results in an error message

Java beta ▶ Run Copy

Hello.java

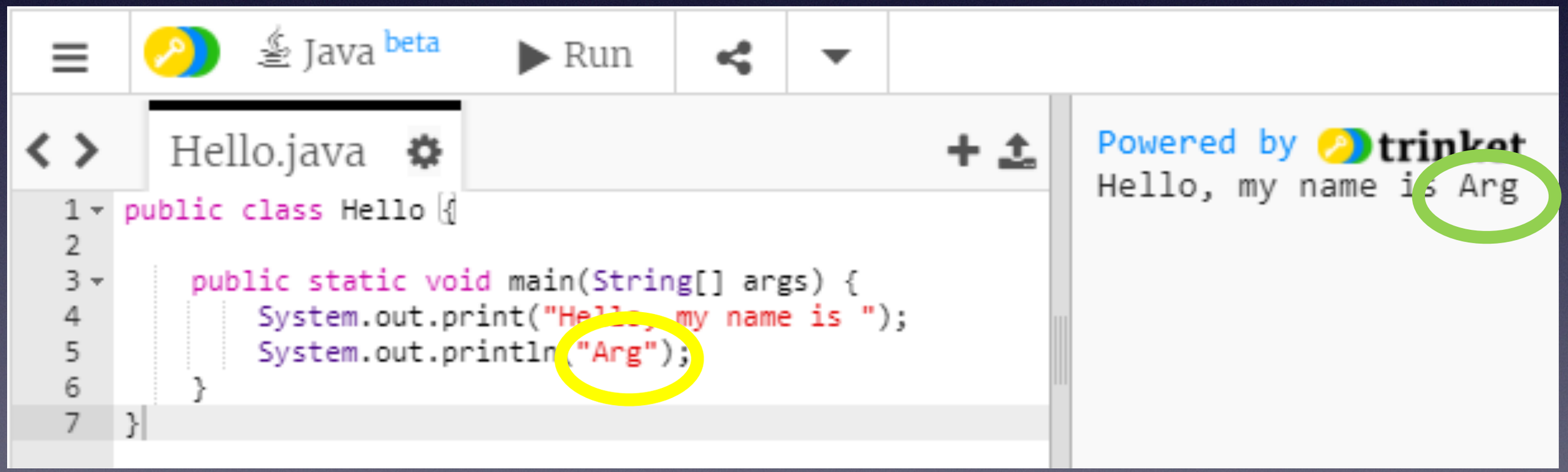
```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.print("Hello, my name is ");
5          System.out.println(Art);
6      }
7  }
```

Powered by trinket

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

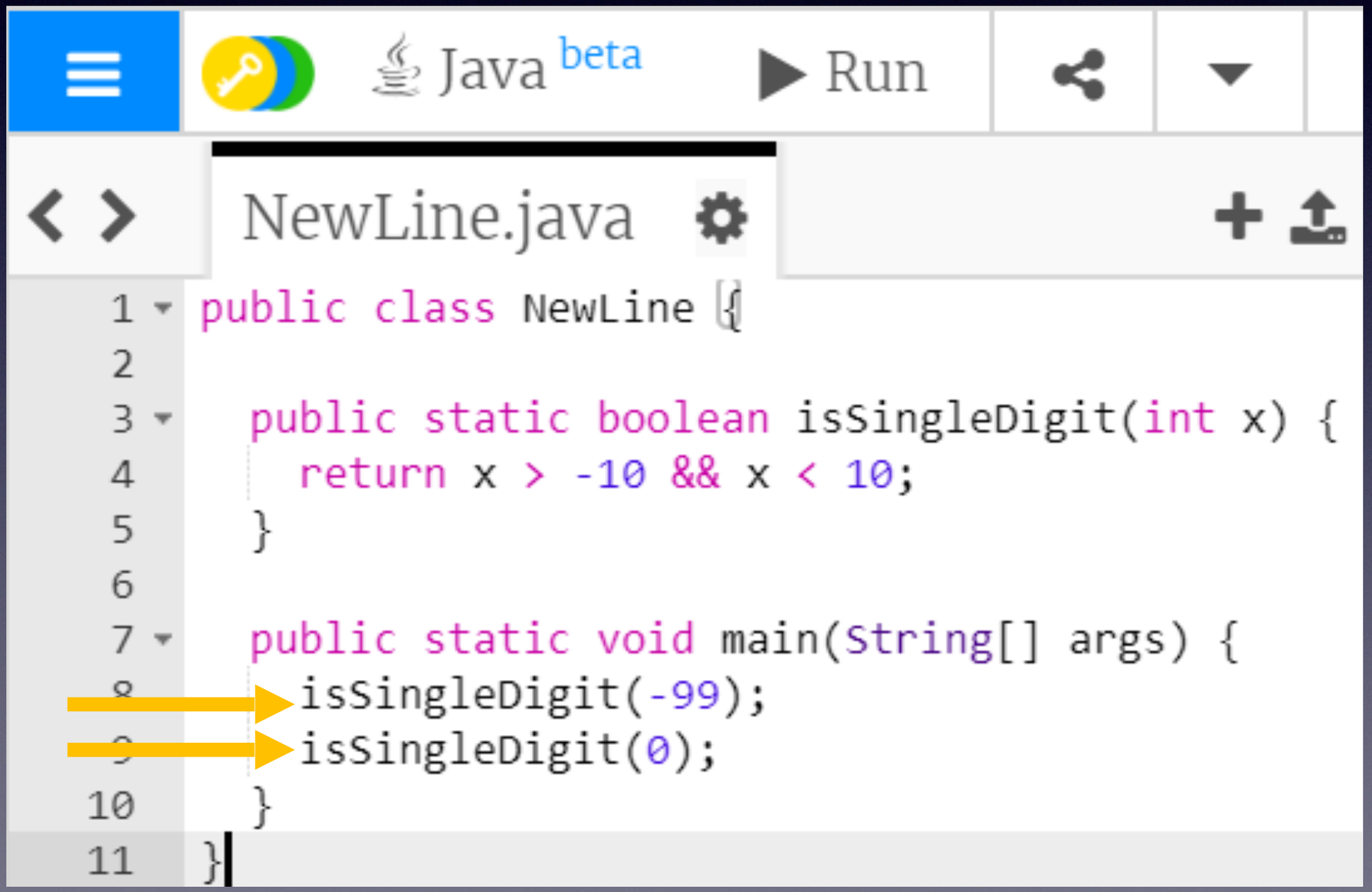
Review: Logic error

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



Will this cause an error message? Is it a mistake?

- Try experimenting



Write a `isDivisible()` function

- Hint: use modulus

Java beta ▶ Run

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isDivisable(int n, int m) {
4      //your code here
5    }
6
7    public static void main(String[] args) {
8      System.out.println(isDivisable(10,2));
9      System.out.println(isDivisable(10,3));
10     System.out.println(isDivisable(12,3));
11     System.out.println(isDivisable(12,5));
12   }
13 }
```

```
Powered b
true
false
true
false
```

Write `isTriangle()` Function

- If any of the three lengths is greater than the sum of the other two, you cannot form a triangle.

Java beta ▶ Run

NewLine.java

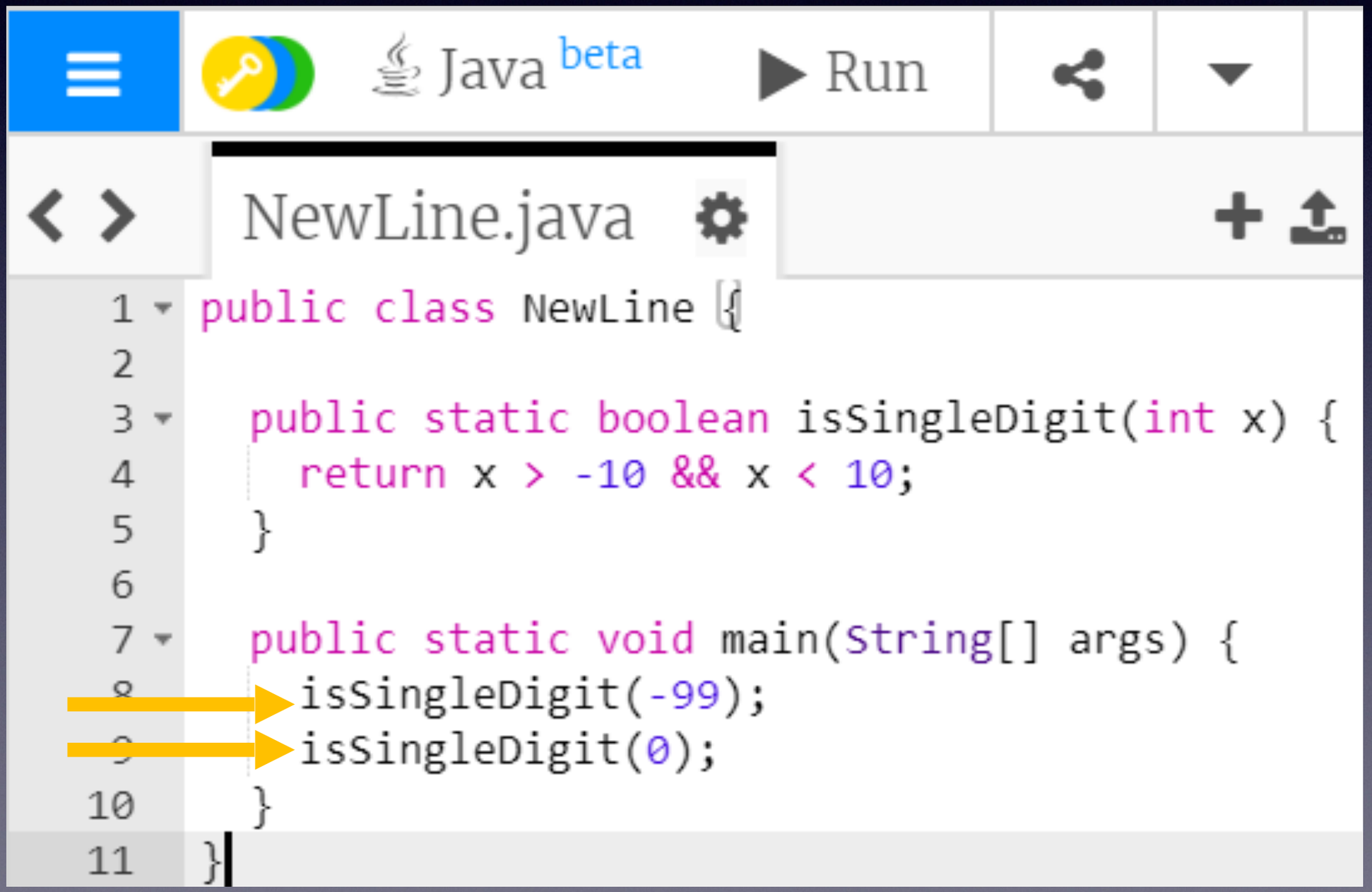
```
1  public class NewLine {
2
3    public static boolean isTriangle(int a, int b, int c) {
4      //your code here
5    }
6
7    public static void main(String[] args) {
8      System.out.println(isTriangle(3,4,5));
9      System.out.println(isTriangle(3,4,8));
10     System.out.println(isTriangle(12,3,11));
11     System.out.println(isTriangle(12,3,5));
12   }
13 }
```

```
Powered b
true
false
true
false
```

Calling functions with return values

1. What happens if you invoke (call) a value method and don't do anything with the result?

A. No error message is shown, but it is still a mistake to call a function that returns a value and not use the returned value



void Functions will not "evaluate"

2. What happens if you use a void function as part of an expression?

A. You get an error message that **`'void' type is not allowed here`**

NewLine.java

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

Powered by trinket

```
NewLine.java:8: error: 'void' type not allowed here
        System.out.println(one() + 7);
                           ^
1 error
```

void Functions

What if the **`one()`** function returned an **`int`**, would that still cause an error message?

A. No

NewLine.java

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

Powered by trinket

```
8
```

String return Functions

What if the **`one()`** function returned an **`String`**, would that still cause an error message?

A. No

NewLine.java

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

Powered by trinket

```
one7
```

Write a `isDivisible()` function

- Hint: use modulus

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isDivisable(int n, int m) {
4      if(n % m == 0)
5        return true;
6      else
7        return false;
8    }
9
10   public static void main(String[] args) {
11     System.out.println(isDivisable(10,2));
12     System.out.println(isDivisable(10,3));
13     System.out.println(isDivisable(12,3));
14     System.out.println(isDivisable(12,5));
15   }
16 }
```

Powered by trinket

```
true
false
true
false
```

Write a `isDivisible()` function

- A better version

NewLine.java

```
1  public class NewLine {
2      public static boolean isDivisable(int n, int m) {
3          if(m == 0)
4              return false;
5          else if(n%m==0)
6              return true;
7          else
8              return false;
9      }
10
11     public static void main(String[] args) {
12         System.out.println(isDivisable(10,2));
13         System.out.println(isDivisable(10,0));
14         System.out.println(isDivisable(10,3));
15     }
16 }
```

```
Powered b
true
false
false
```

Write `isTriangle()` Function

- If any of the three lengths is greater than the sum of the other two, you cannot form a triangle.

NewLine.java

```
1  public class NewLine {
2
3    public static boolean isTriangle(int a, int b, int c) {
4      if(c > a + b)
5        return false;
6      else if(b > a + c)
7        return false;
8      else if(a > b + c)
9        return false;
10     else
11       return true;
12   }
13
14   public static void main(String[] args) {
15     System.out.println(isTriangle(3,4,5));
16     System.out.println(isTriangle(3,4,8));
17     System.out.println(isTriangle(12,3,11));
18     System.out.println(isTriangle(12,3,5));
19   }
20 }
```

Powered by trinket

```
true
false
true
false
```