

Computer Programming in Java

Bush School *CPJava Fall 2022*

Block D

Welcome to CPJava!



Dru & not Gru!

Chandru Narayan
Bicyclist Astronomer Engineer
Teaching Astronomy and CS



Introductions!

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

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

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

Course Requirements

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

Expectations of Learning

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

CPJava course is online!

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

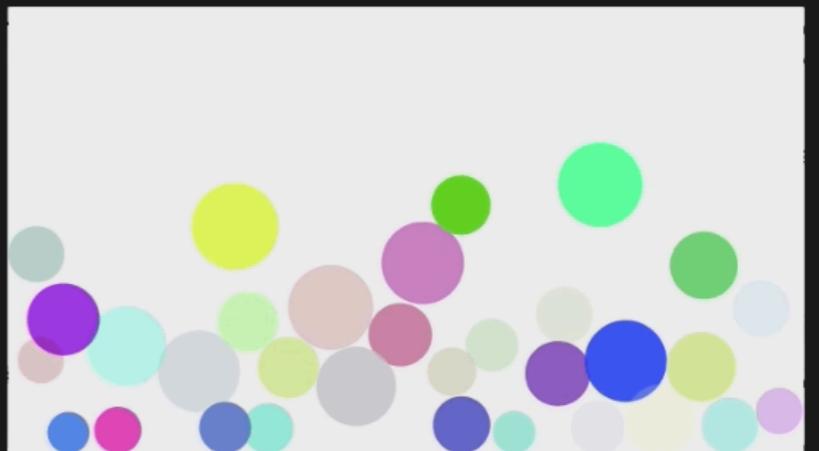
You are here! Click here to access.

Bookmark this
You will need it every day!



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

Bush School CPJava Fall Semester 2020



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

CPJava - Computer Programming in Java Course

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

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

Course Schedule

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

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

Grading Policy

Points are assigned for:

Completing Classwork Exercises

Submitting Projects

Professionalism & Integrity

AP CSA exam is on May 3rd 2023

Talk to me if you are going to take it!

Grades

Grades are assessed each Semester

50% Projects

(Includes Final Project or APCSA Exam)

35% Classwork / Assignments

15% Student Portfolio

Details are available
at the CPJava Course website

How to succeed

Simply write lots of Java code

Publish your work to the web (Github)

You cannot learn programming by reading or
watching!

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

Working cooperatively with your peers - Pair
Programming!

Don't be afraid to try new things!

Professionalism

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

Office Hours and Class Assistance

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

Tools/Resources for CPJava

CPJava website - Lesson Plans and Projects

Bush Portal - Course Syllabus, Schedule and Grades

CPJava Google Classroom - You should have received an invitation

CPJava Slack - You should have received an invitation

Applications for Laptop

- Github, Processing, Visual Code

- See Instructions in Google Classroom

CSAwesome Online

- Online Textbook, Lessons, Exercises, Detailed Reference

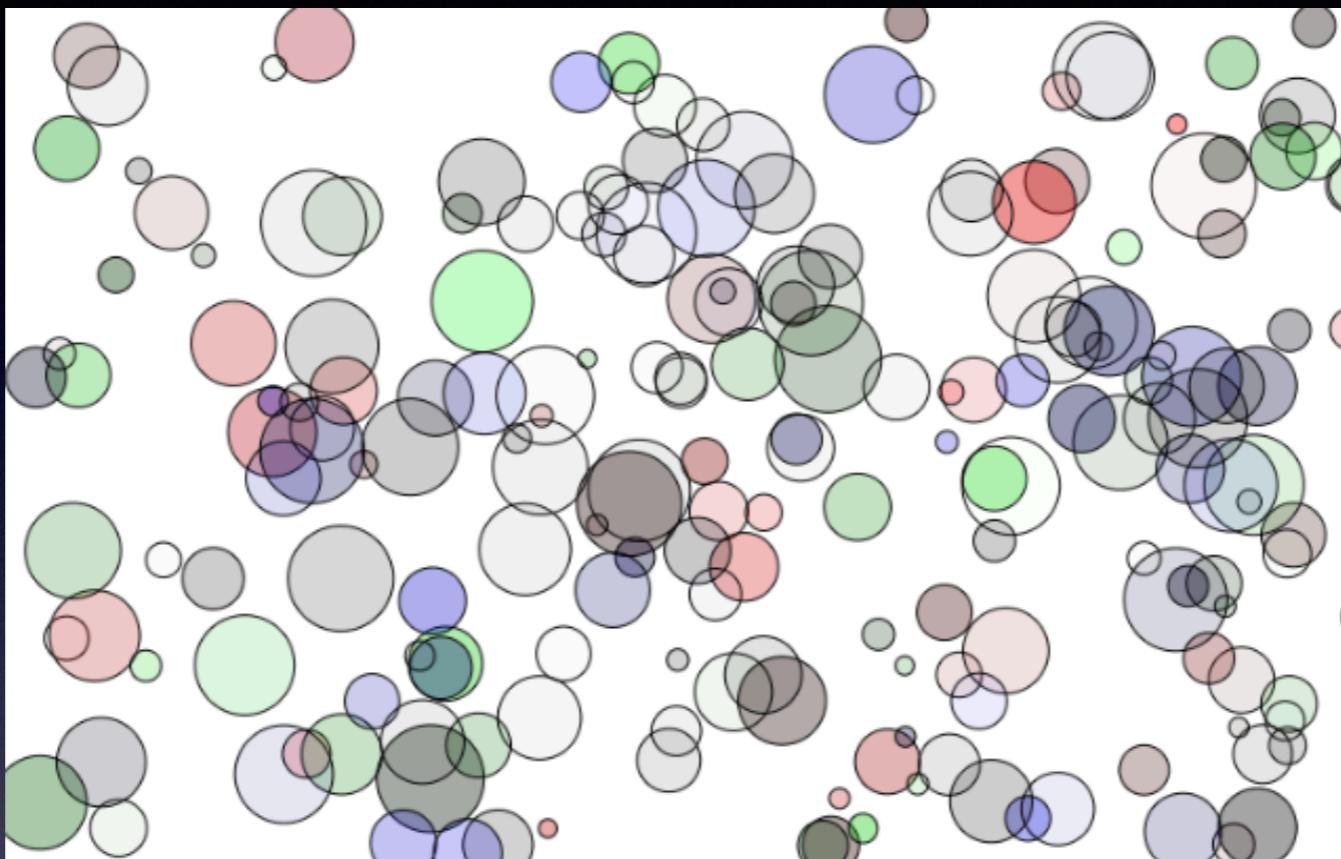
System Requirements

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

Sufficient disk space, a functioning laptop battery
fully charged!

Functioning Camera and Microphone - especially
required for Paired Programming

Let's run our first Java Program!



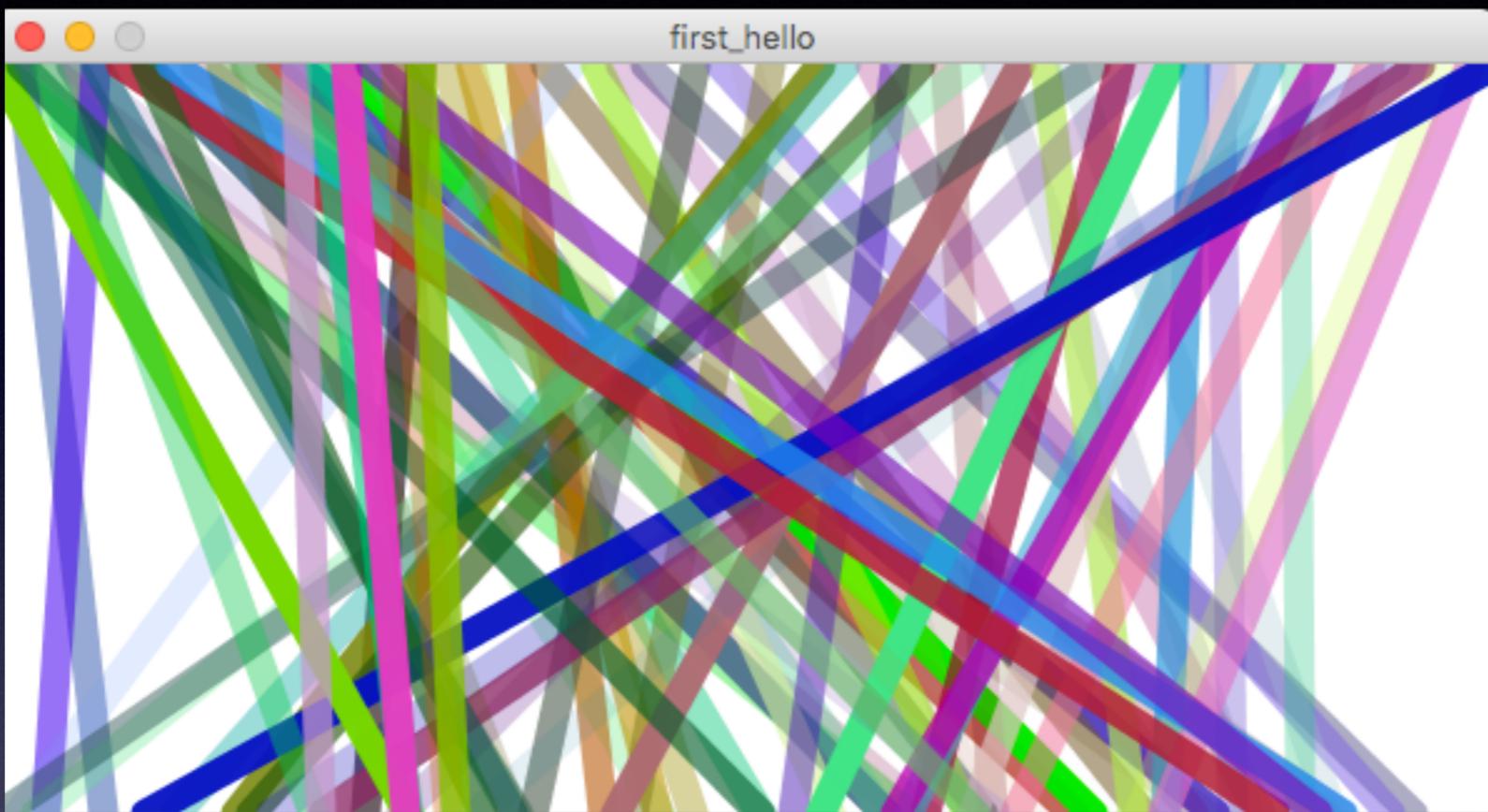
Bubbles - click to run the program

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

Can you freeze all the bubbles?

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

Let's modify our first Java Program!



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

Java Basics

Unit 0

Unit 0

The common building blocks in programming languages
are:

Variables

Loops

if statements

Functions (aka “methods”)

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

A basic Java Hello program

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



The screenshot shows a Java code editor interface. At the top, there's a navigation bar with icons for file operations, the 'trinket' logo, a 'Java beta' button, a 'Run' button, a 'Share' button, and a dropdown menu. Below the bar, the code editor displays a file named 'ABasicJavaProgram.java'. The code itself is:

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

A basic Java program

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



The screenshot shows a Java code editor interface. At the top, there's a navigation bar with icons for file operations, the word "trinket", a Java logo, and buttons for "Run" and "Share". Below the bar, the title "ABasicJavaProgram.java" is displayed. The code area contains the following Java code:

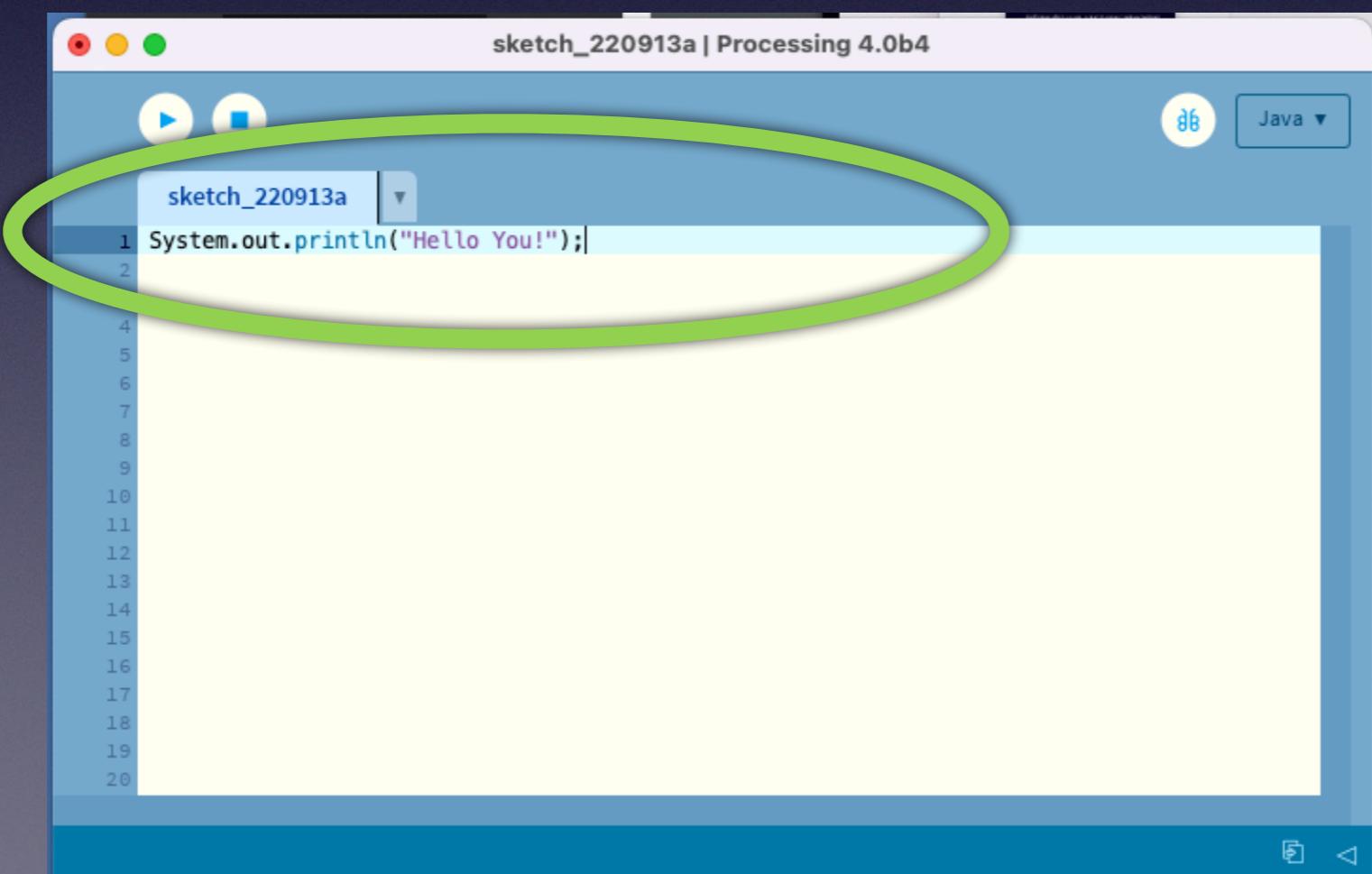
```
1 public class ABasicJavaProgram
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello You!");
6     }
7 }
```

The line "System.out.println("Hello You!");" is highlighted with a large green oval.

A basic Java program in Processing



Much easier!



sketch_220913a | Processing 4.0b4

```
sketch_220913a
1 System.out.println("Hello You!");
```

The screenshot shows the Processing IDE interface. The title bar reads "sketch_220913a | Processing 4.0b4". Below the title bar is a toolbar with a play button, a stop button, and a "Java" dropdown menu. The main workspace contains a code editor with the following content:

```
sketch_220913a
1 System.out.println("Hello You!");
```

The line "1 System.out.println("Hello You!");" is highlighted with a green oval. The code editor has a light blue background and white text. The numbers 1 through 20 are visible on the left side of the code editor, likely representing line numbers. The bottom of the screen shows the operating system's dock with icons for Finder, Mail, Safari, and others.

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



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

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

A green oval highlights the line "System.out.println("Hello You!");". A yellow arrow points from the bottom right towards the end of this line.

String

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



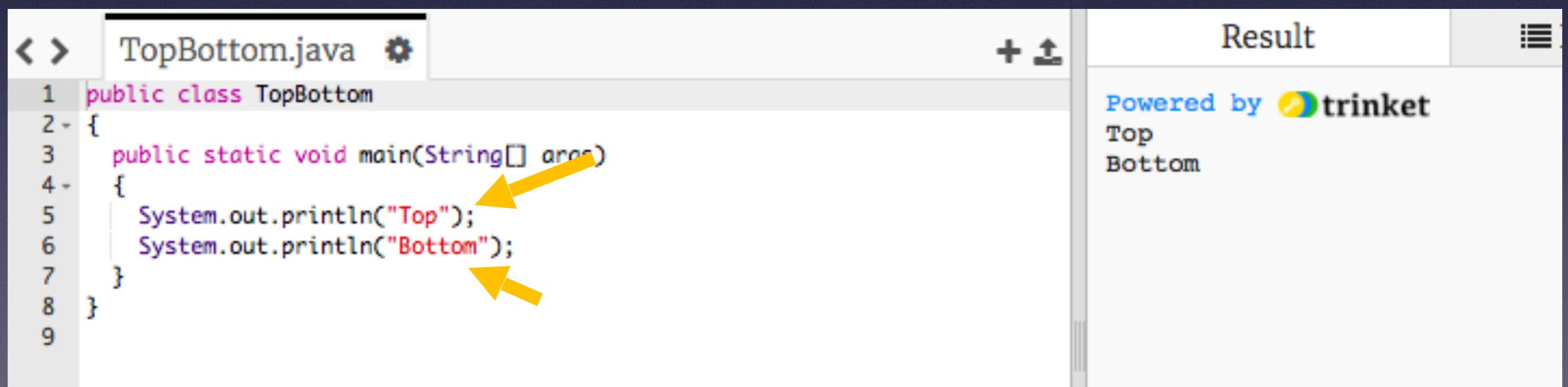
The screenshot shows a Java code editor interface from trinket. The title bar says "trinket Java beta". The file name is "ABasicJavaProgram.java". The code is:

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

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

print() vs println()

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



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

Code:

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

Result:

Powered by trinket

Top
Bottom

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

print() vs println()

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



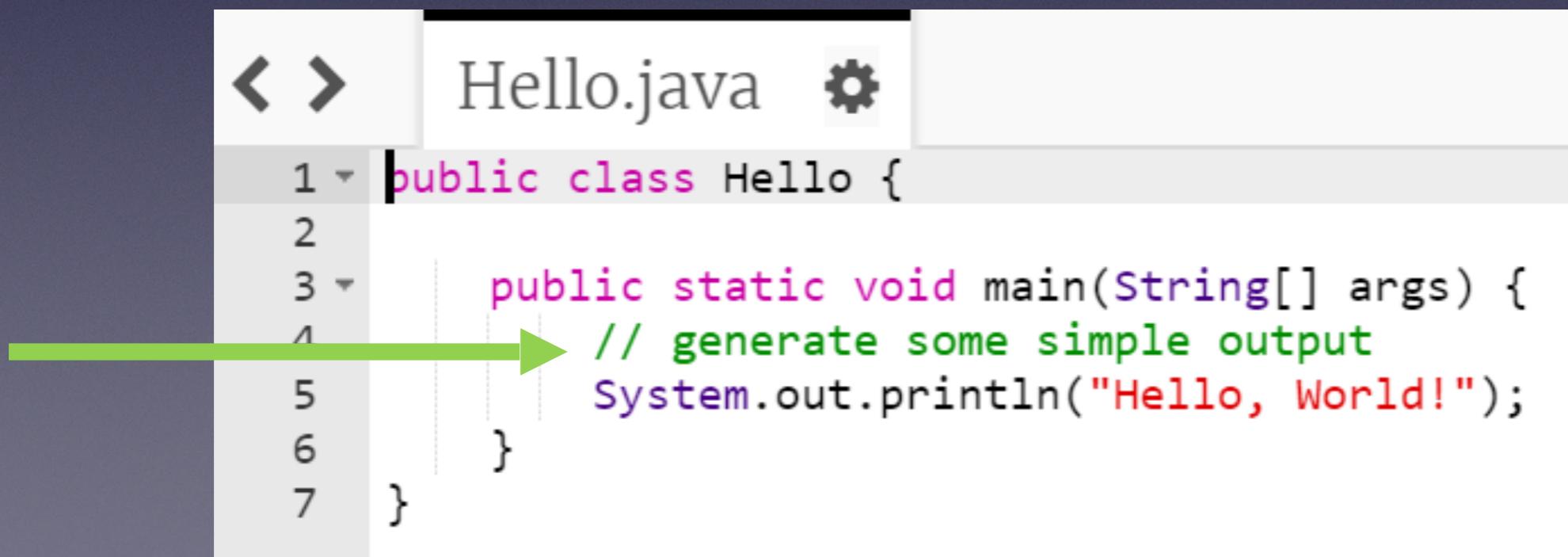
```
Goodbye.java
public class Goodbye {
    public static void main(String[] args) {
        System.out.print("Top");
        System.out.println("Bottom");
    }
}
```

Powered by trinket

TopBottom

Comments

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



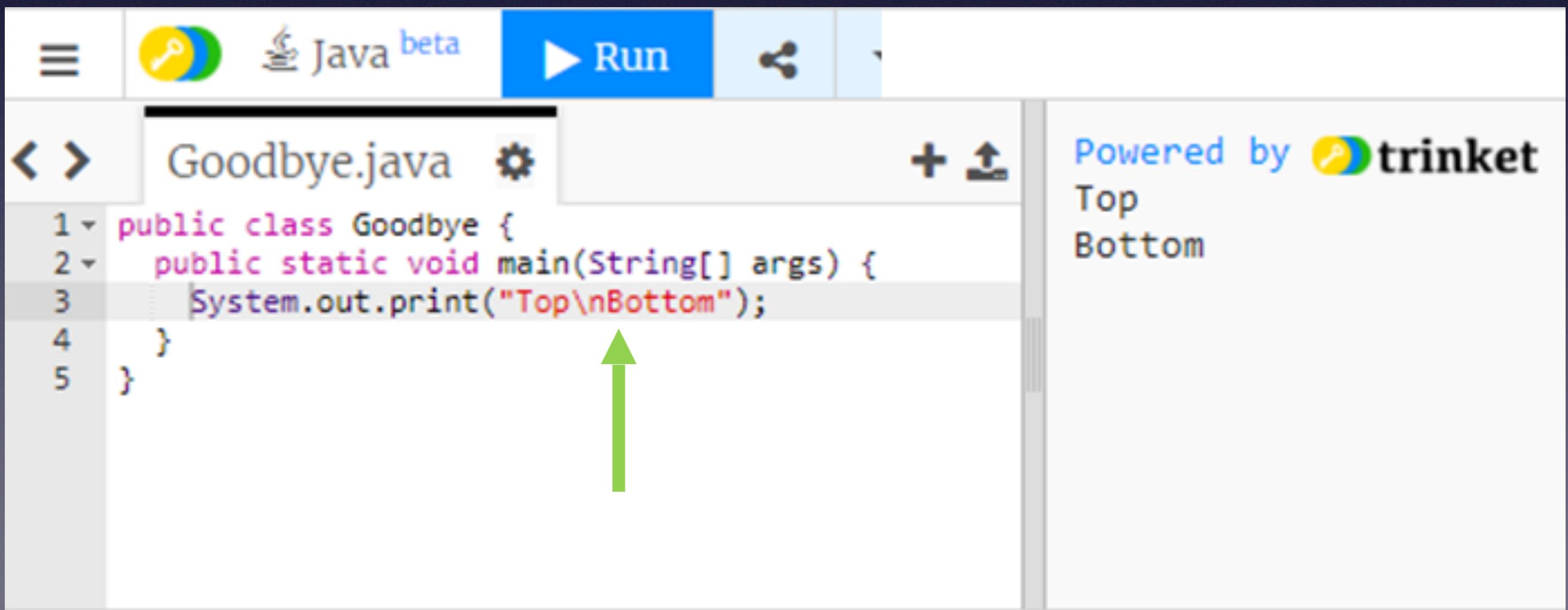
```
1 < > public class Hello {  
2  
3     public static void main(String[] args) {  
4         // generate some simple output  
5         System.out.println("Hello, World!");  
6     }  
7 }
```

Escape Sequences

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

Common Java Escape Sequences

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



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

Powered by  trinket
Top
Bottom

Common Java Escape Sequences

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

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

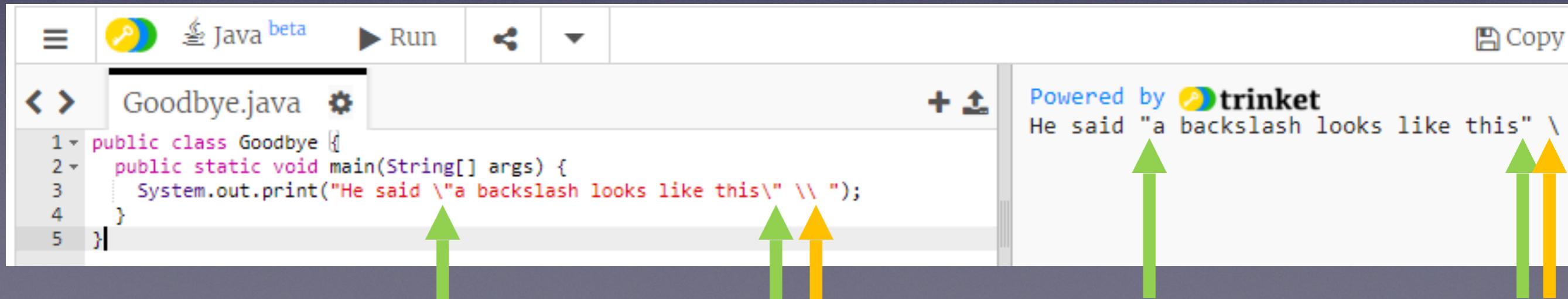
- Toolbar:** Includes icons for file operations, a key icon, "Java beta", "Run", and other development tools.
- File List:** Shows "Goodbye.java" as the active file.
- Code Editor:** Displays the following Java code:

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

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

Common Java Escape Sequences

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



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

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

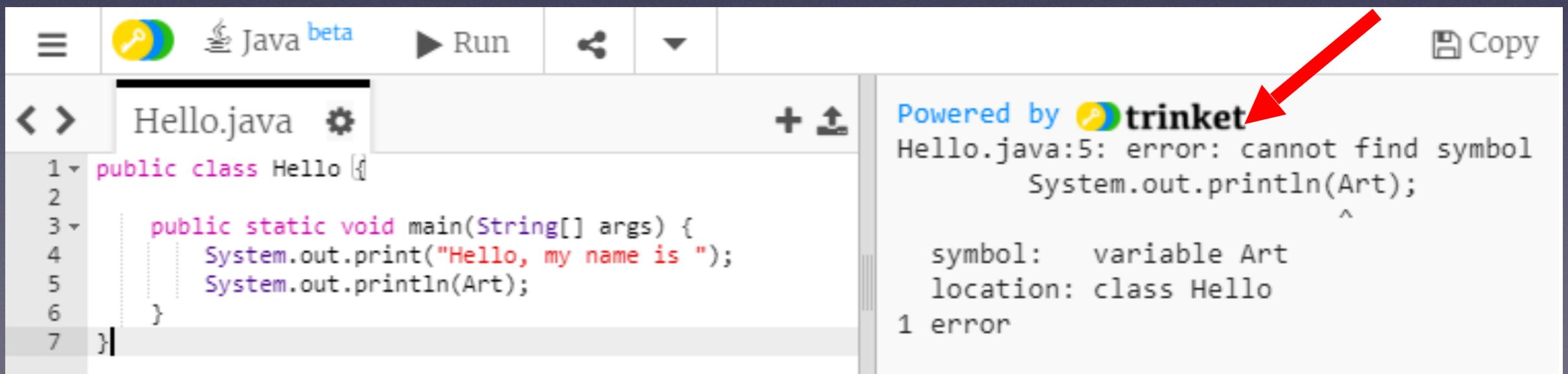
The output window shows the result of the `System.out.print` statement: "He said "a backslash looks like this"" followed by a backslash character. Three green arrows point from the three backslash characters in the code to the corresponding characters in the output. A yellow arrow points from the double quotes in the code to the double quotes in the output.

Debugging

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

Errors

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



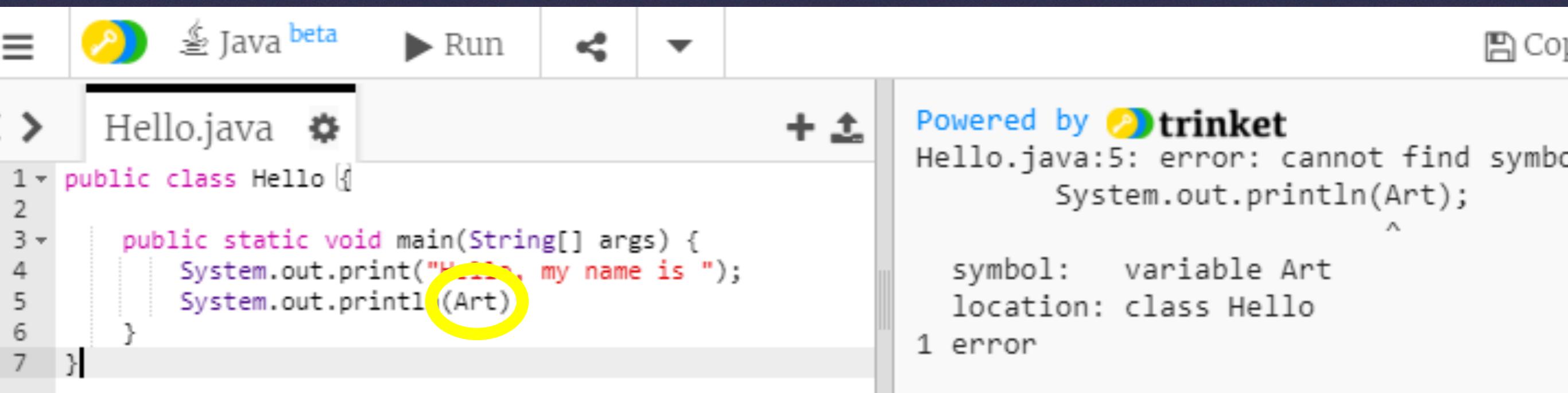
The screenshot shows a Java code editor interface. The top bar includes a key icon, a Java beta logo, a Run button, and a Copy button. The main area displays a file named "Hello.java" with the following code:

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

The status bar at the bottom right shows an error message: "Powered by trinket" followed by "Hello.java:5: error: cannot find symbol System.out.println(Art);". Below this, it says "symbol: variable Art" and "location: class Hello". It also indicates "1 error". A red arrow points from the left towards the error message in the status bar.

Syntax error

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



The screenshot shows a Java code editor interface. The top bar includes icons for file operations, a Java logo, and tabs for "Java beta". Below the tabs, a file named "Hello.java" is selected. The code editor displays the following Java code:

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

A yellow circle highlights the word "Art" in the line "System.out.println(Art);". To the right of the editor, the output window shows the error message:

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 code editor interface. The top bar includes a logo, the text "Java beta", and a "Run" button. The code editor window displays a file named "Hello.java". The code contains a main method that prints two strings to the console. A yellow circle highlights the misspelling "Arg" in the second print statement. To the right, the output window shows the program's execution: "Hello, my name is Arg", with the word "Arg" circled in green.

```
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 }
```

Powered by trinket
Hello, my name is Arg

(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 code editor interface. The top bar includes icons for file, run, and copy, along with a Java beta logo. The left sidebar lists files: Hello.java. The main editor area contains the following code:

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

A yellow circle highlights the line `System.out.println(1/0);`. A red oval encircles the error message displayed in the terminal window on the right:

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

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 a Java development environment with the following interface elements:

- Top bar: Includes a key icon, the text "Java beta", a "Run" button, and other standard IDE icons.
- Project navigation: Shows "Hello.java" selected.
- Code editor: Displays the following Java code:

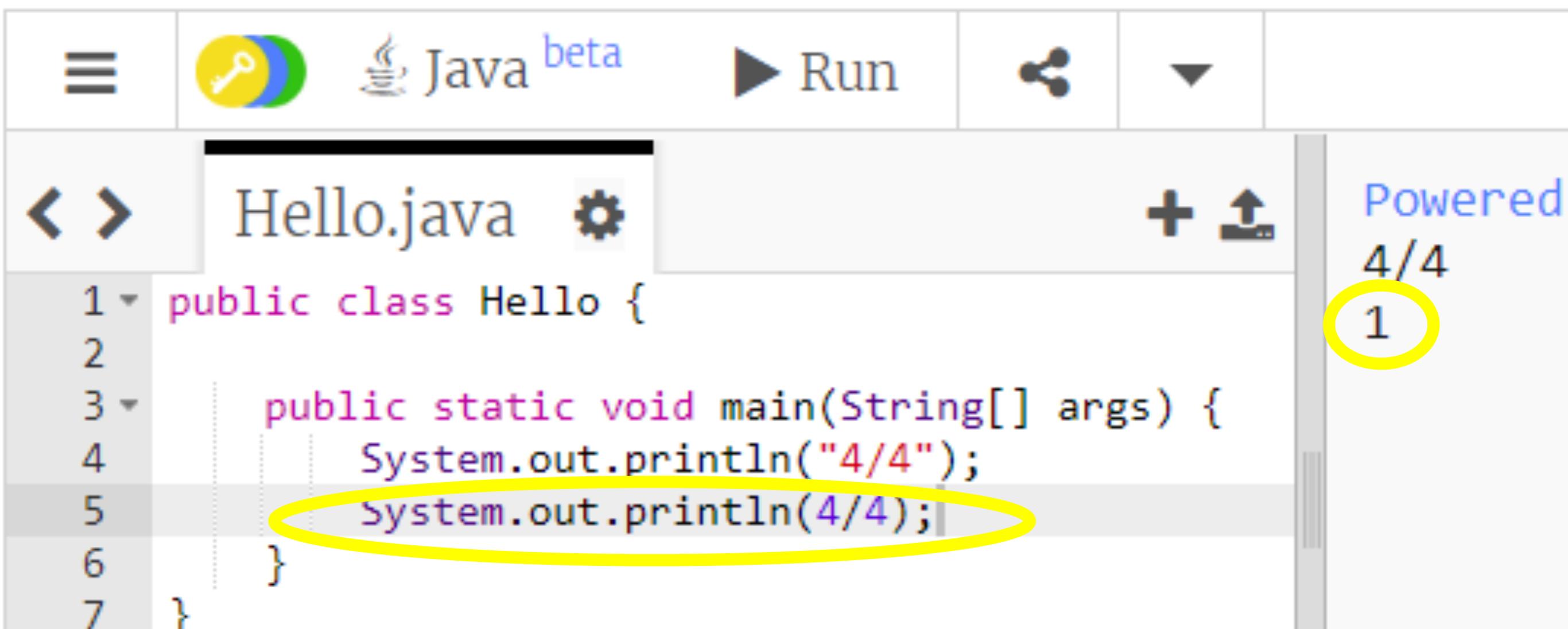
```
1 public class Hello {  
2  
3     public static void main(String[] args) {  
4         System.out.println("4/4");  
5         System.out.println(4/4);  
6     }  
7 }
```
- Output panel: Shows the printed output:

Powered
4/4
1

The output "4/4" is circled in red.

Literals vs. Expressions

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



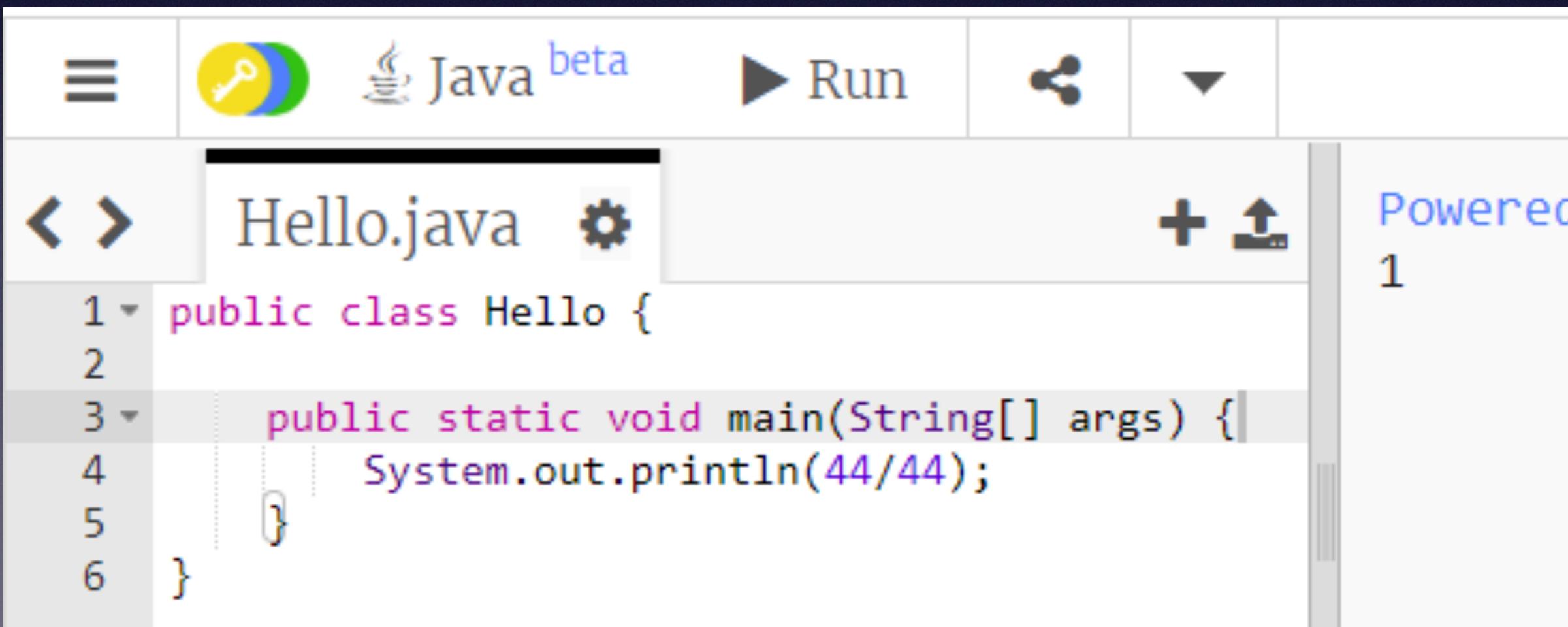
```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("4/4");  
        System.out.println(4/4);  
    }  
}
```

Four 4s challenge

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

Four 4s challenge

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



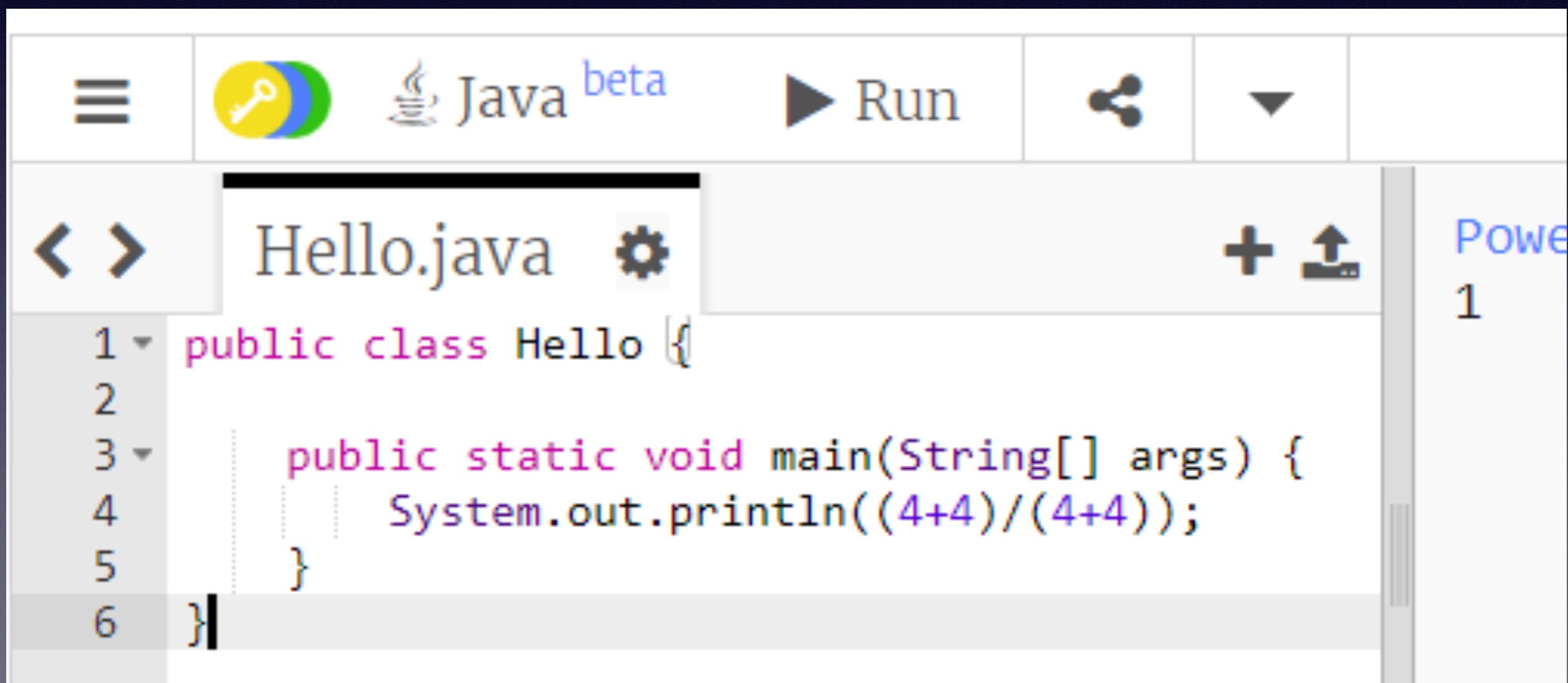
The screenshot shows a Java code editor interface. The top bar includes icons for file, run, and share, along with the text "Java beta". Below the bar, a file named "Hello.java" is selected. The code editor displays the following Java code:

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

The code prints the value 1 to the console.

Four 4s challenge

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



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

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

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

Unit 1A

Java Functions and Parameters

Functions (“methods”)

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

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

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

no argument



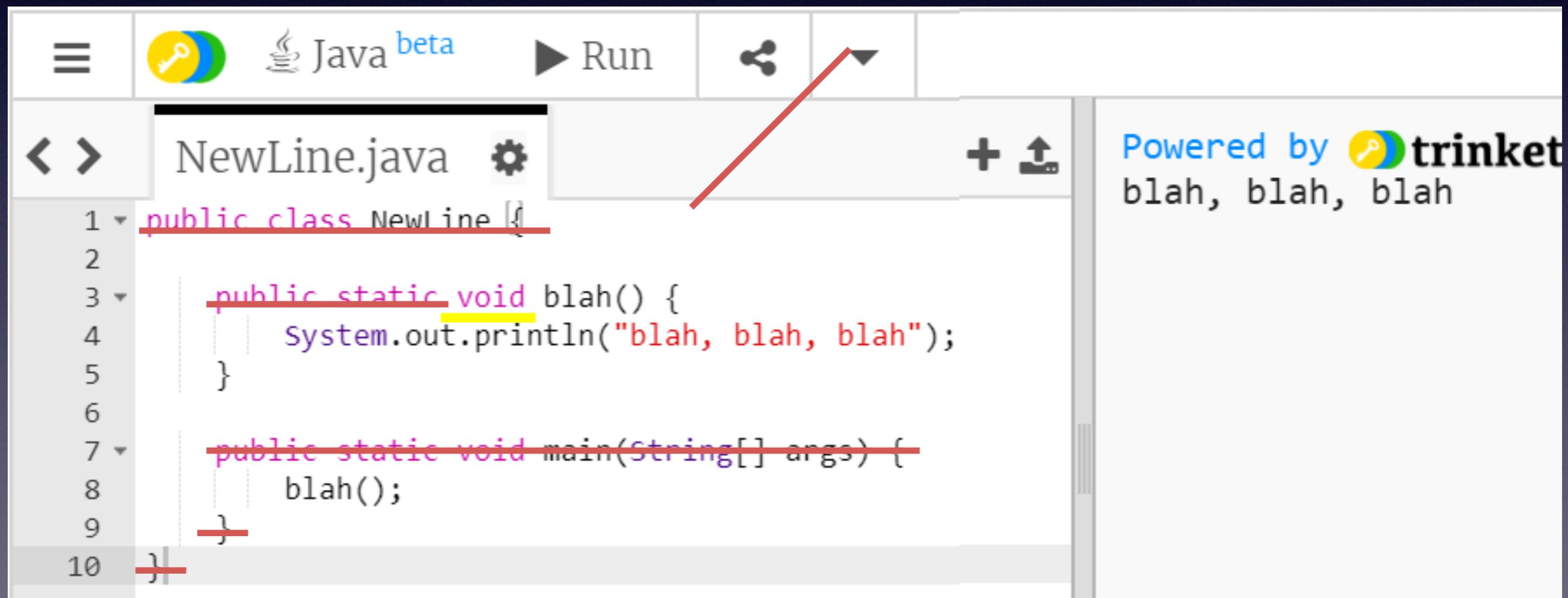
argument

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 *values*

void functions (aka methods)

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



A screenshot of a Java code editor interface. The title bar says "Java beta". The main area shows a file named "NewLine.java". The code contains a class definition with a constructor and a method named "blah". A red arrow points from the text "the crossed out items!" in the previous slide to the word "void" in the method signature "public static void blah()".

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

Powered by trinket
blah, blah, blah

void functions (aka methods)

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

The screenshot shows the Processing IDE interface with the title bar "sketch_220913a | Processing 4.0b4". The code editor contains the following Java code:

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

void setup() {
    blah();
}

void draw() {
```

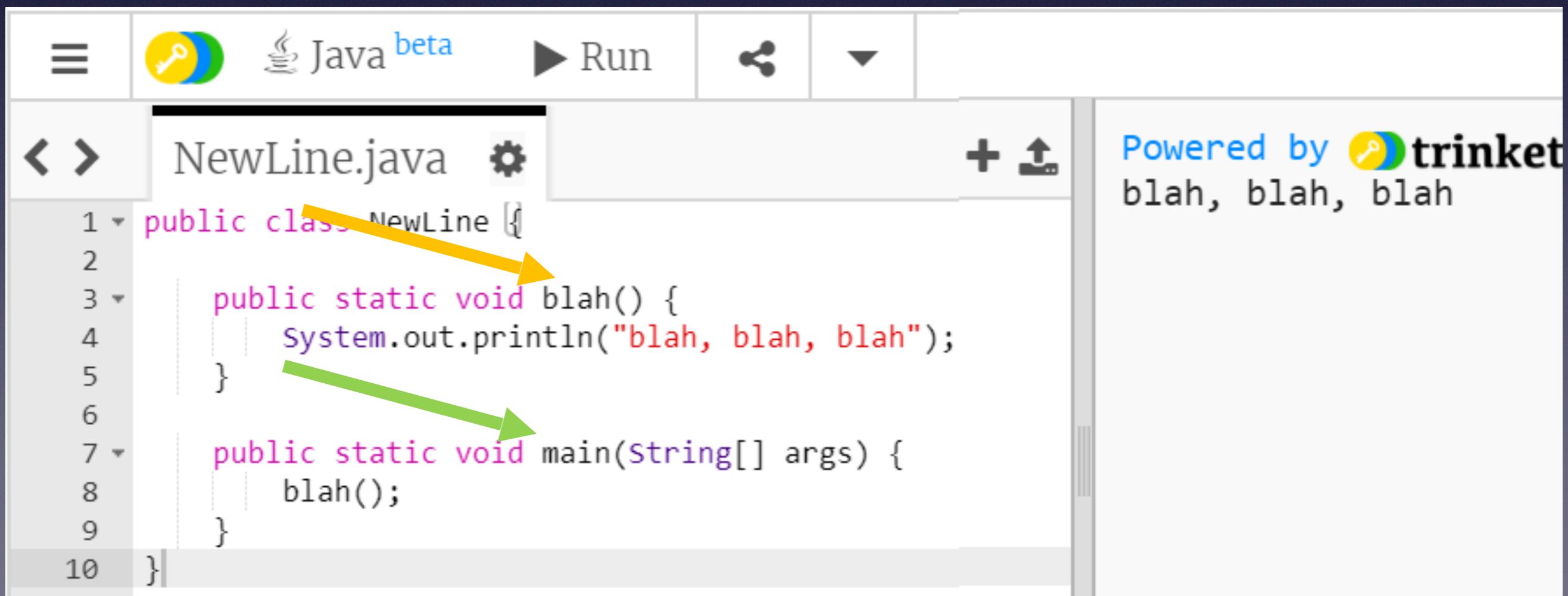
Annotations with red arrows and text boxes explain the code:

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

The status bar at the bottom displays the text "blah, blah, blah".

void functions (aka methods)

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



```
NewLine.java
public class NewLine {
    public static void blah() {
        System.out.println("blah, blah, blah");
    }
    public static void main(String[] args) {
        blah();
    }
}
```

Powered by trinket
blah, blah, blah

void functions (aka methods)

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

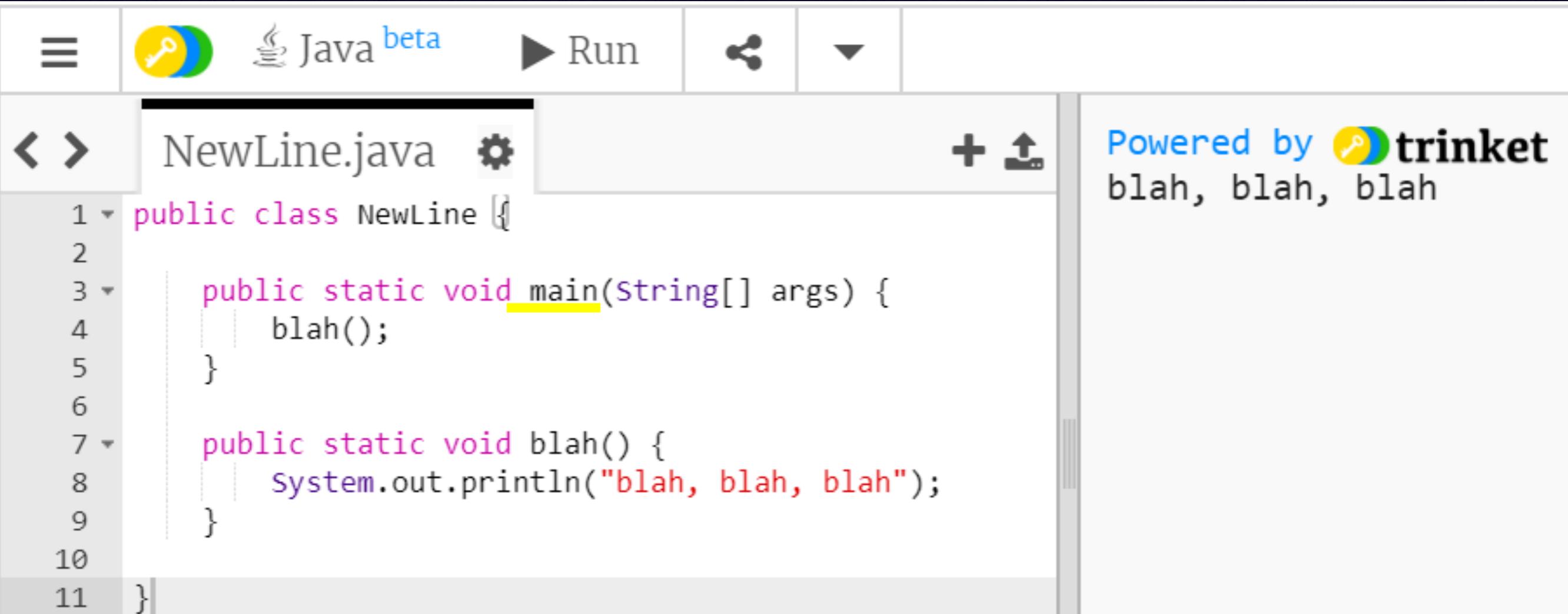
The screenshot shows a Java code editor interface. The top bar includes a file icon, the text "Java beta", a "Run" button, and other standard icons. Below the bar, the code editor displays a file named "NewLine.java". The code contains two methods: a static void method named "blah" and a main method that calls "blah". The "blah" method prints the string "blah, blah, blah" to the console. The "main" method also prints "blah, blah, blah". The code is numbered from 1 to 10. A yellow highlight is placed under the word "blah" in the "main" method's call to the "blah" method. The right side of the editor shows the output of the code execution, which is "blah, blah, blah".

```
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



The screenshot shows a Java code editor interface. The top bar includes a key icon, the text "Java beta", a "Run" button, and other standard toolbar items. The left sidebar lists files: "NewLine.java" is selected and shown in the main pane, while "Output" is also listed. The right sidebar displays the output of the program: "Powered by trinket" followed by the text "blah, blah, blah". The code itself is as follows:

```
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 }
```

void functions (aka methods)

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

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

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

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

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**

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

- Toolbar:** Includes icons for file operations, a key icon, "Java beta", "Run", and a share icon.
- Code Area:** The file "PrintTime.java" is open. The code defines a class `PrintTime` with a `printTime` method and a `main` method. The `printTime` method takes two integer parameters: `hour` and `minute`. The `main` method declares two integers `h` and `m`, and calls the `printTime` method with these values as arguments.
- Annotations:** Two green arrows point from the parameter declarations in the `printTime` method signature down to the corresponding variable declarations in the `main` method. Two orange arrows point from the argument values `h` and `m` in the `printTime` call up to the parameter declarations in the `printTime` method signature.
- Status Bar:** Shows "Powered by" and the time "7:35".

```
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 }
```

Arguments must match Parameters in number, order and type

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

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

- Toolbar:** Includes icons for file operations, Java beta, Run, and a share button.
- Code Editor:** A tab labeled "Main.java" is selected. The code defines a class Main with a main method and a someFunction method. The someFunction method takes two parameters: an int and a String. In the main method, the arguments are swapped: a String is passed first, followed by an int. Two green arrows point from the word "String" in the function call to the parameter declaration "int num", and two yellow arrows point from the word "int" in the function call to the parameter declaration "String word".
- Output Panel:** Labeled "Powered by trinket". It displays a compilation error message: "Main.java:9: error: incompatible types: String cannot be converted to int someFunction(w,n); ^". A note below says: "Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output".

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
public class Main {
    public static void someFunction(int num, String word){
        System.out.println(n + ", " + w);
    }
    public static void main(String[] args) {
        int n = 1;
        String w = "one";
        someFunction(n,w);
    }
}
```

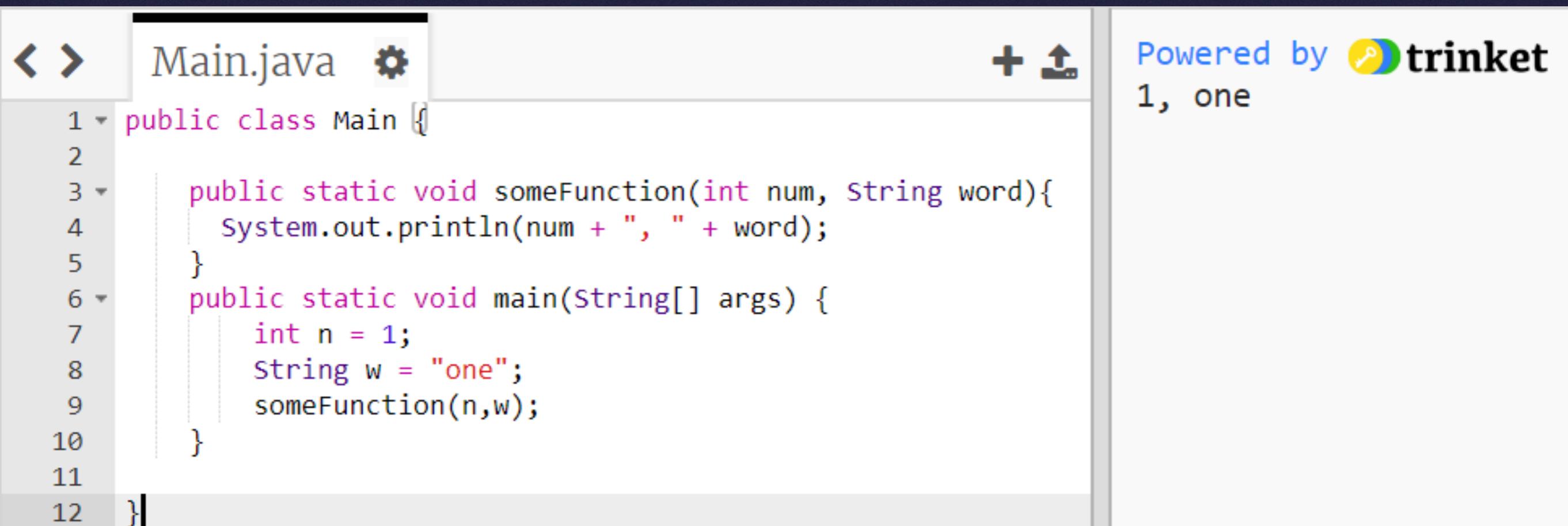
Powered by trinket

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

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

Now the program is correct

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



Main.java

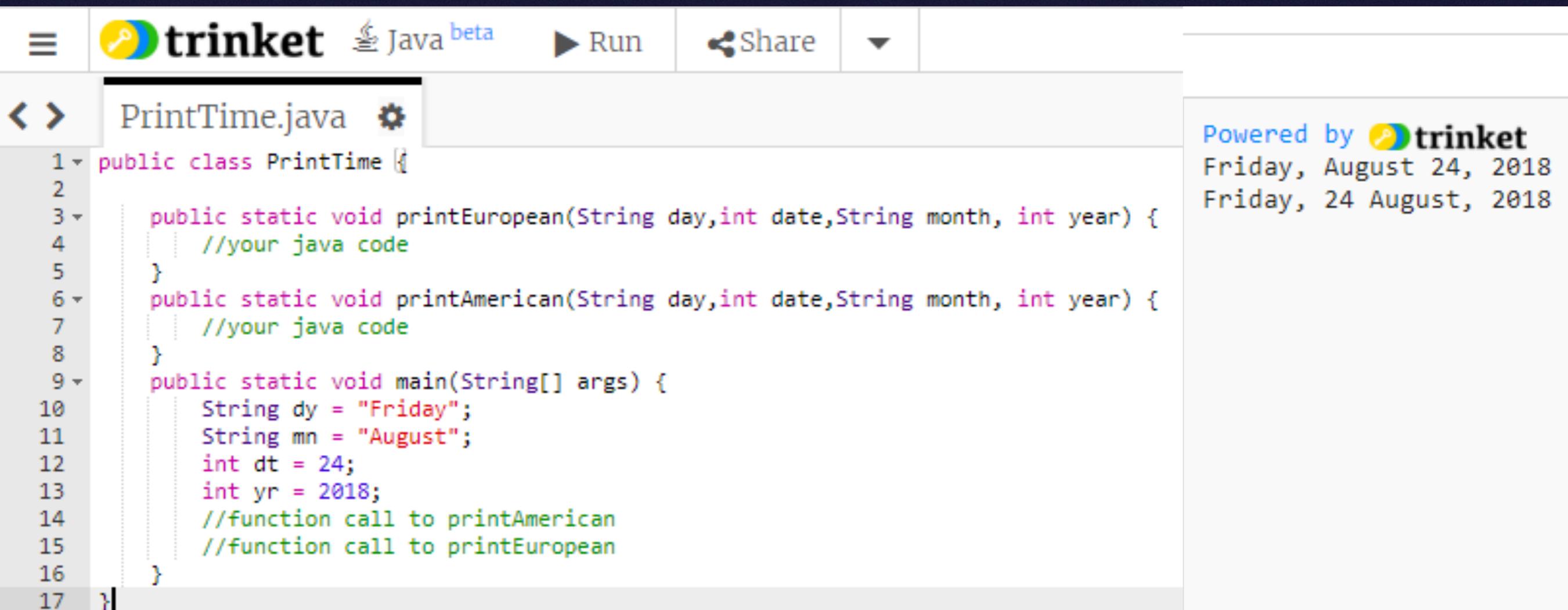
```
1 public class Main {  
2  
3     public static void someFunction(int num, String word){  
4         System.out.println(num + ", " + word);  
5     }  
6     public static void main(String[] args) {  
7         int n = 1;  
8         String w = "one";  
9         someFunction(n,w);  
10    }  
11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

Powered by trinket

1, one

PrintTime Function Assignment

Complete the PrintTime Function Assignment
from the Functions and Parameters section



The screenshot shows a Java code editor interface for the trinket platform. The title bar indicates it's Java beta. The code editor window has tabs for 'PrintTime.java' and a gear icon. The code itself is as follows:

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

To the right of the code editor, there is a sidebar with the text "Powered by trinket" and two dates: "Friday, August 24, 2018" and "Friday, 24 August, 2018".

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



The screenshot shows the Trinket Java beta interface. The code editor window contains a Java file named PrintTime.java with the following code:

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

A yellow arrow points to the "Link" option in the "Share" dropdown menu, which is highlighted in gray.

- Twitter
- Facebook
- Google+
- Email
- Link**
- Embed
- Download

Unit 1B

Variables Types and Operators

Unit 1B

- 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



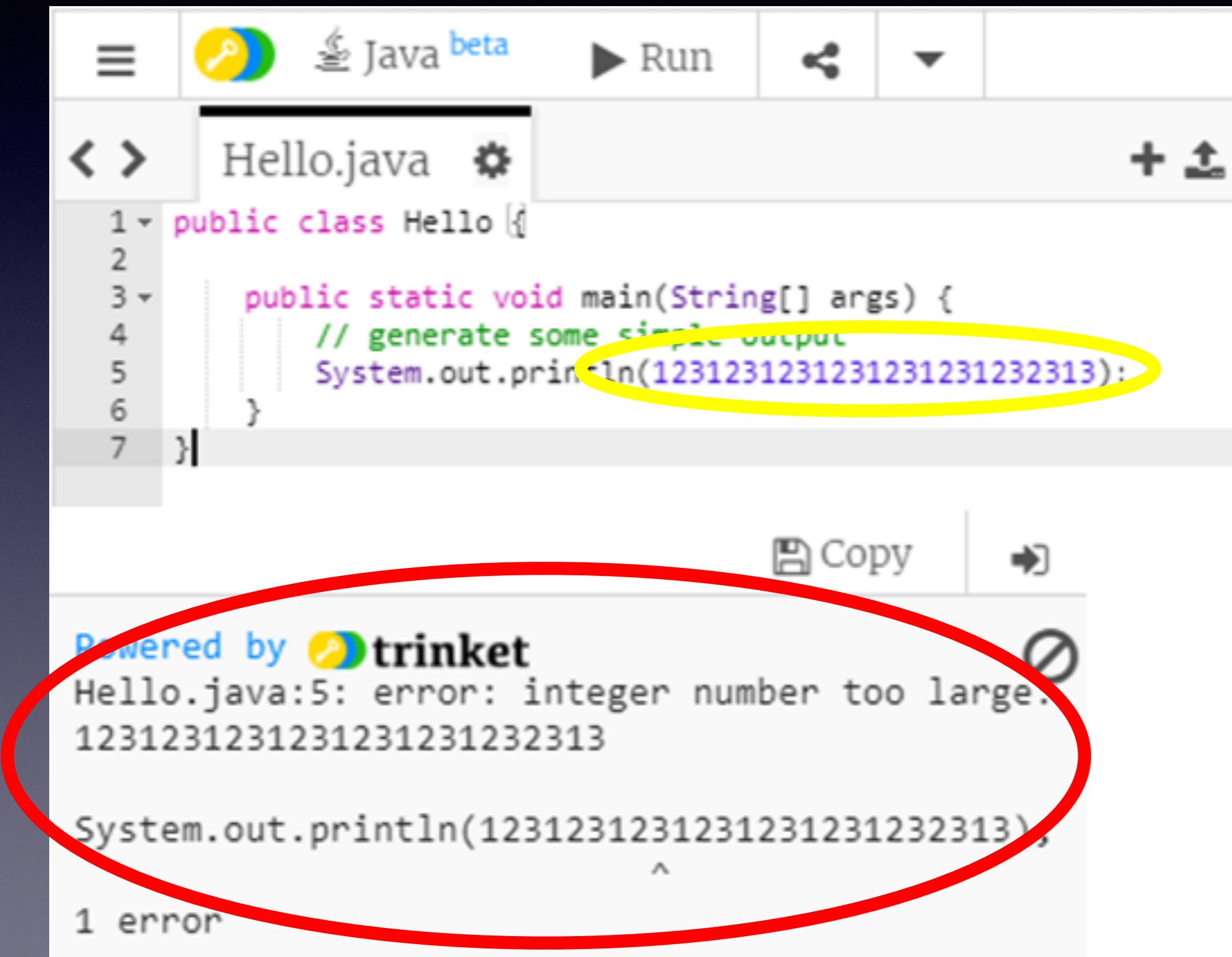
Variables: Parking space analogy

- 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



```
public class Hello {  
    public static void main(String[] args) {  
        // generate some simple output  
        System.out.println(1231231231231231231232313);  
    }  
}
```

Powered by  trinket

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

System.out.println(1231231231231231231232313),
^

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 code editor interface with the following details:

- Toolbar:** Includes icons for file operations, a key icon, "Java beta", a "Run" button, and share options.
- Code Area:** Displays the file "Hello.java" containing the following code:

```
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 }
```
- Output Area:** Shows the output of the program:

Powered by trinket

0.5555555555555556

0.555556

A yellow arrow points from the first output value to the line `System.out.println(dNum);`. A green arrow points from the second output value to the line `System.out.println((float)dNum);`.

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 ("")



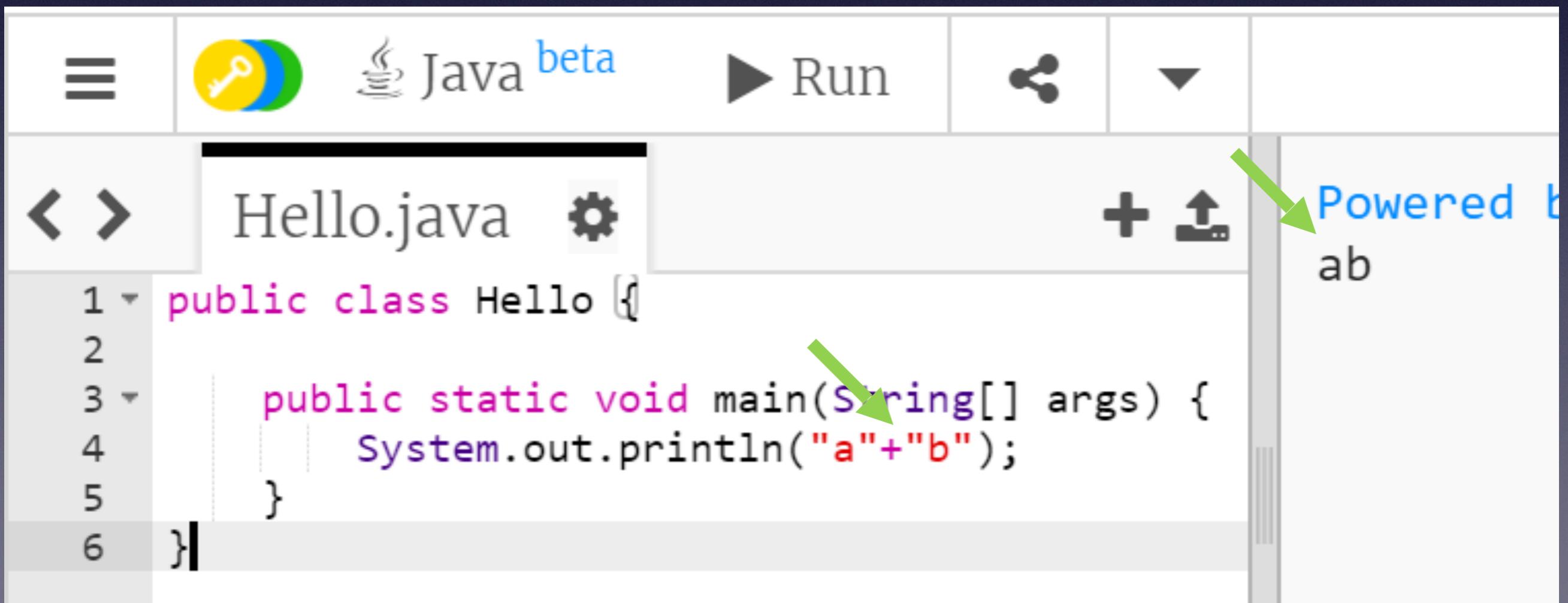
A screenshot of a Java code editor interface. The top bar includes icons for file, run, and share, and displays "Java beta". The left sidebar shows a file named "Hello.java" and a settings gear icon. The main code area contains the following Java code:

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

A green arrow points to the string literal "Hey, give me 5!" in line 4. To the right of the code, there is a preview window showing the output: "Powered by trinket 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



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

+ 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
```

- 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 code editor 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 string literal " \"" in the code, highlighting the empty String used to separate the two division operations. A green arrow points to the output window on the right, which displays the result "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

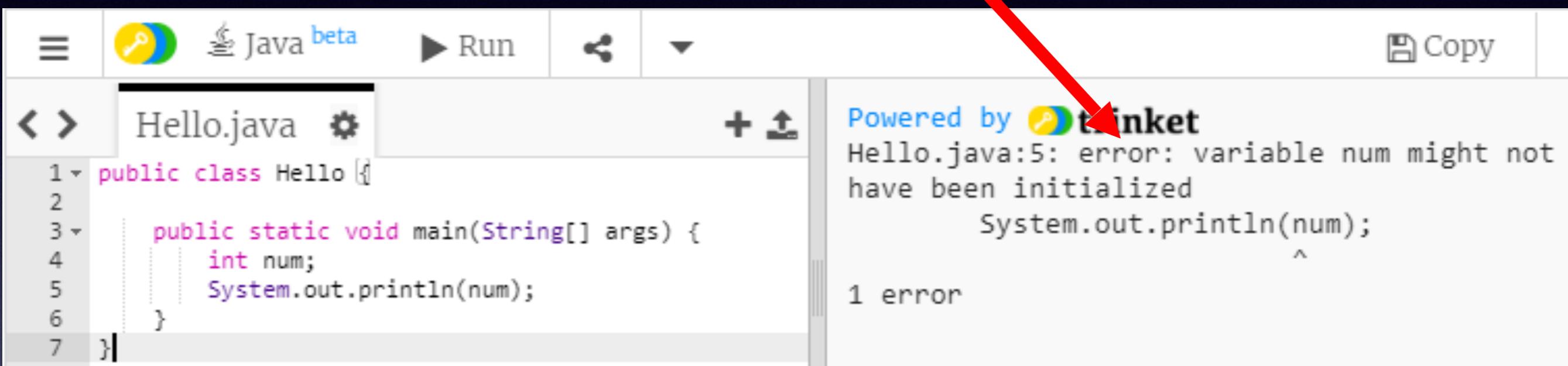
Variable names and keywords

- 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?



A screenshot of a Java code editor interface. The top bar includes a key icon, "Java beta" text, "Run" button, and "Copy" button. The left sidebar shows a file named "Hello.java" with a gear icon. The main code area contains the following Java code:

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

The right side of the interface displays the build output:

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

A large red arrow points from the question "What's the error?" down to the "Tinket" logo in the error message.

Variable initialization

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

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

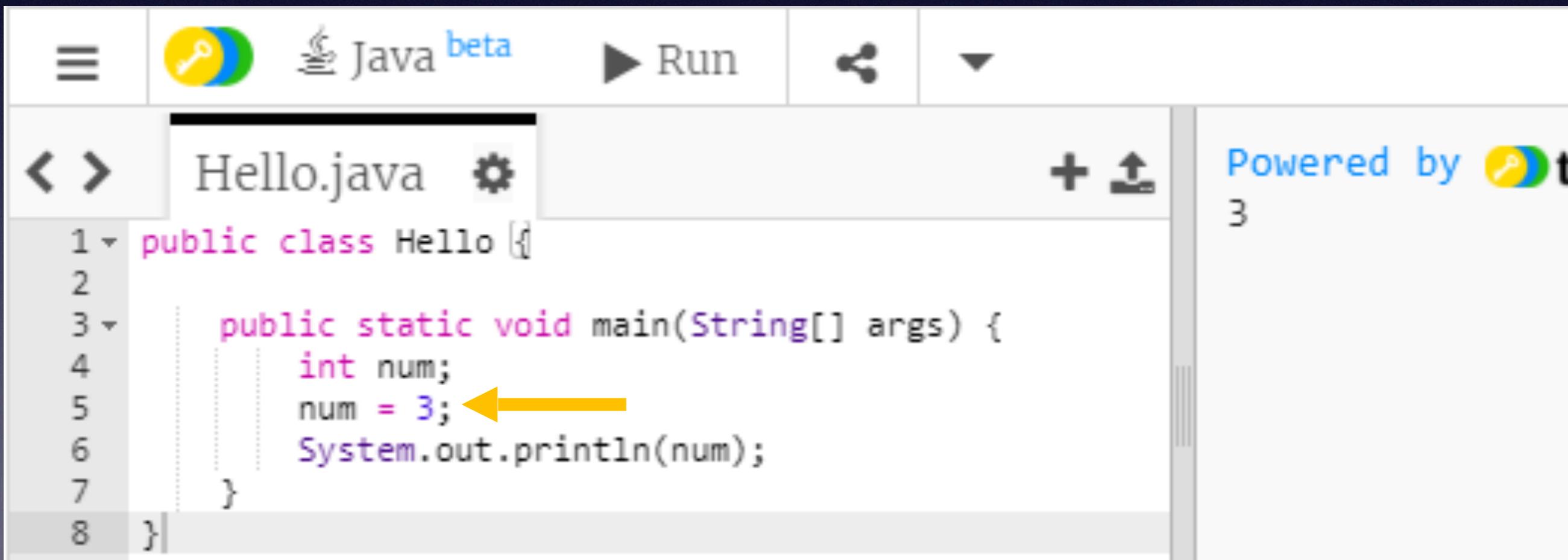
```
1 public class Hello {  
2     public static void main(String[] args) {  
3         int num;  
4         System.out.println(num);  
5     }  
6 }  
7 }
```

To the right of the code, an error message is displayed:

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



```
public class Hello {  
    public static void main(String[] args) {  
        int num;  
        num = 3; ←  
        System.out.println(num);  
    }  
}
```

Declare *and* initialize

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

```
int num = 3;
```

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

Comments

```
//Single Line
```

```
/*
```

```
Multi  
Line
```

```
*/
```

- Tells the computer to ignore some text

Used to:

- Write notes to yourself or other programmers

- Temporarily disable code

Kahoot!

