

# 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!



/ 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
<b>FALL 2021 SEMESTER</b>	<b>12-Weeks</b>	<b>Sep to Dec 2021</b>
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
<b>SPRING 2022 SEMESTER</b>	<b>15-Weeks</b>	<b>Jan to May 2022</b>
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 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 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

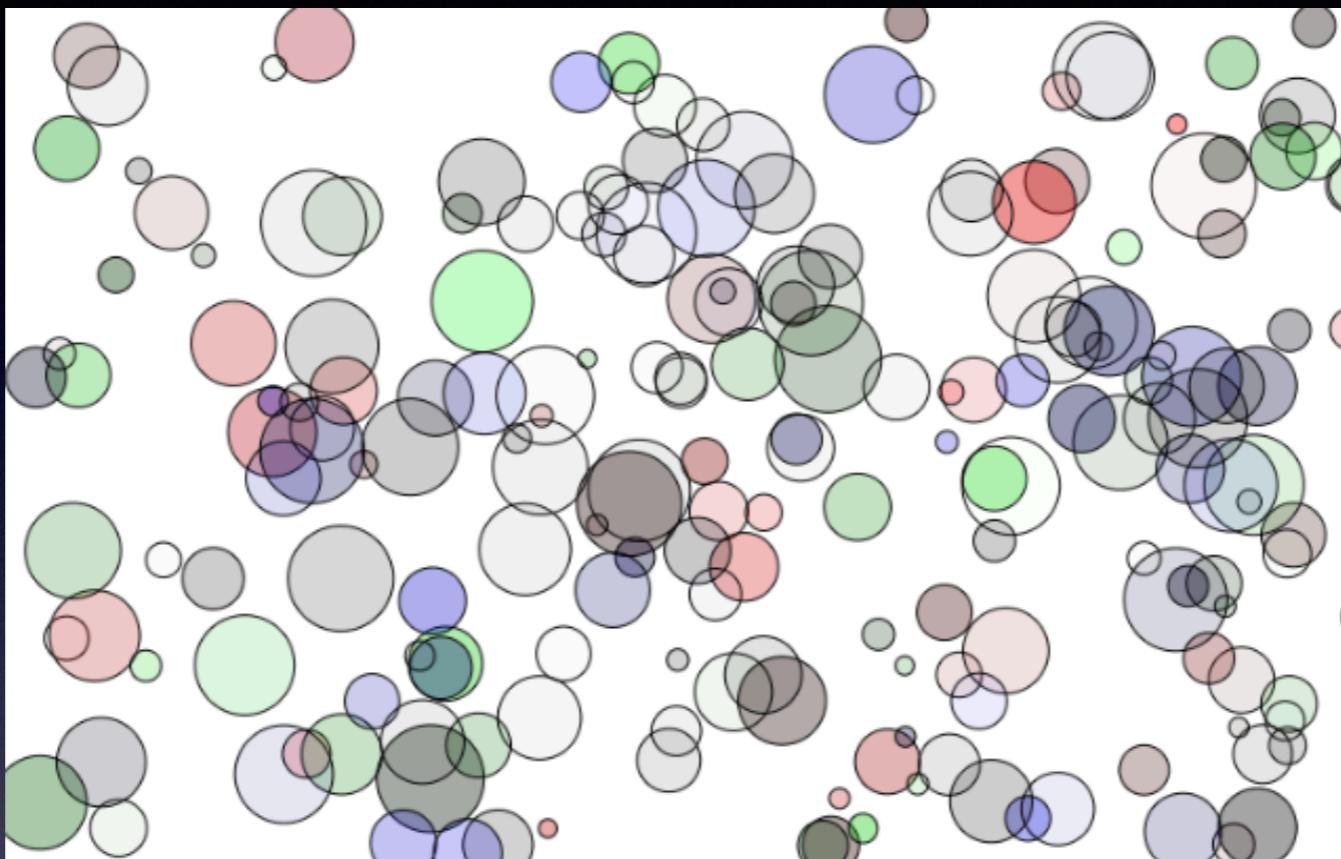
# System Requirements

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!



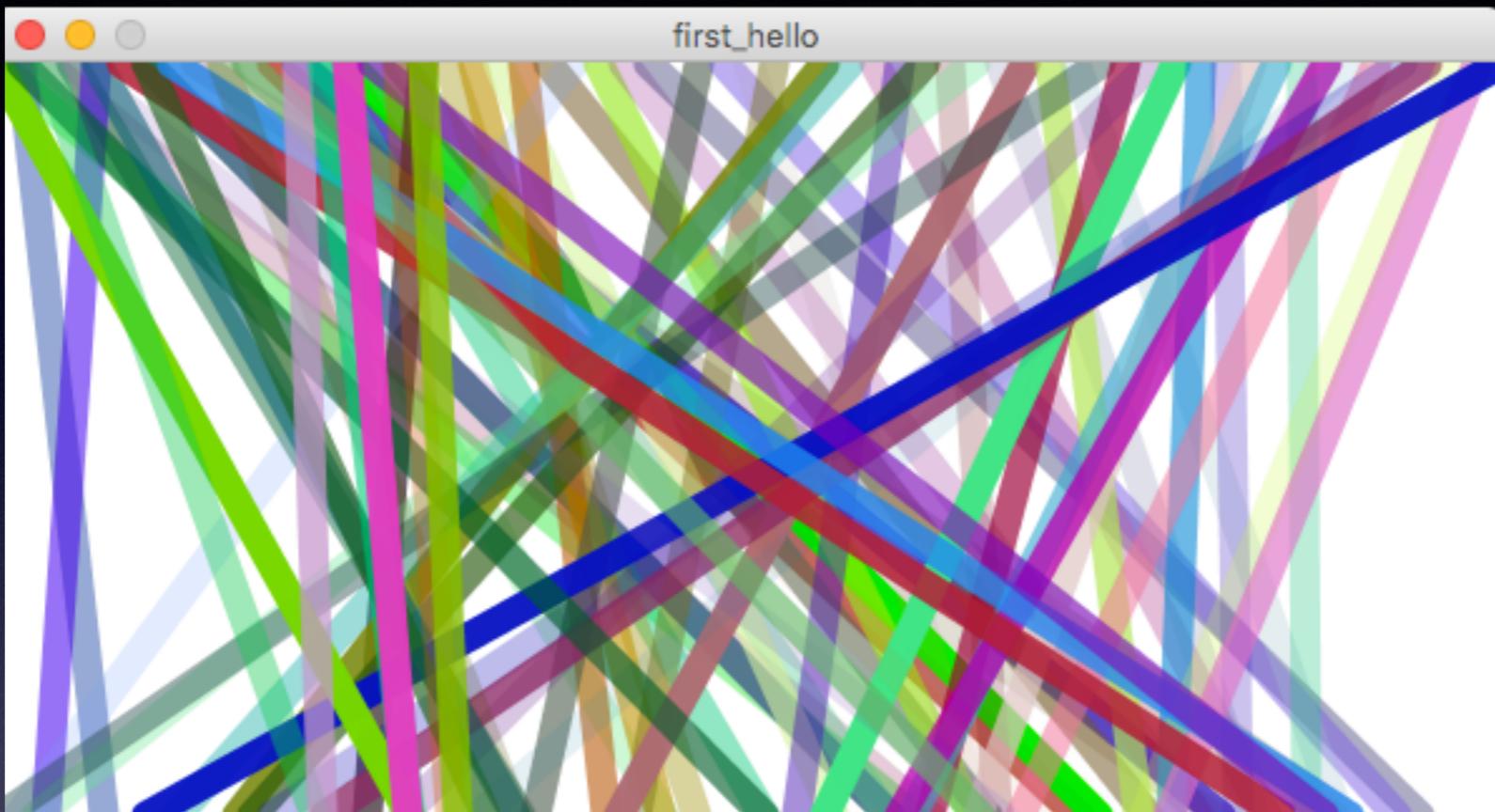
Bubbles - click to run the program

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 . . .



```
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 we can ignore everything except the code circled in green



```
trinket Java beta Run Share ABasicJavaProgram.java
public class ABasicJavaProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hello You!");
    }
}
```

# 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 the trinket website. 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 large 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 ("")



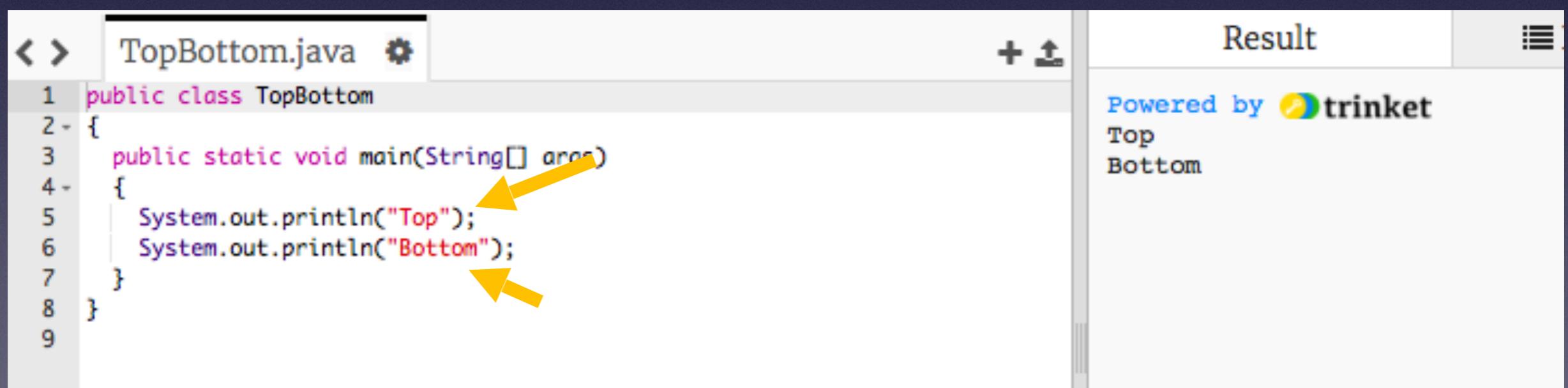
The screenshot shows a Java code editor interface from trinket. The title bar says "trinket Java beta". The file name is "ABasicJavaProgram.java". The code is:

```
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!" in line 5.

# 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"



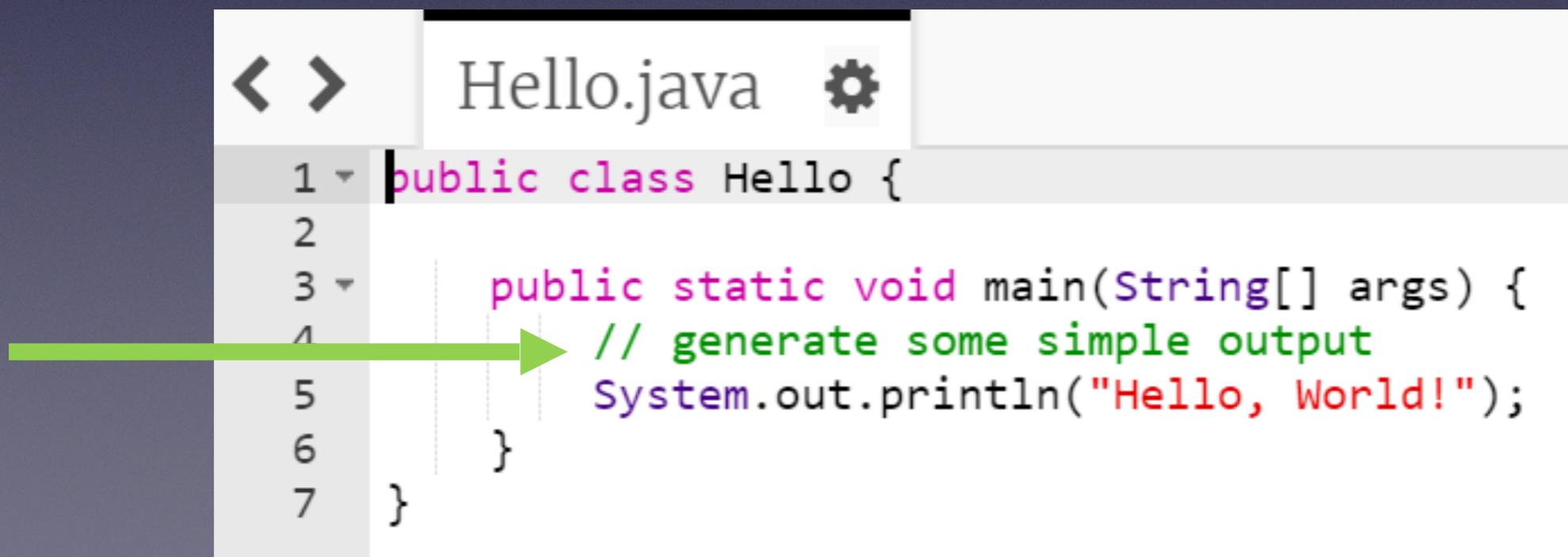
The screenshot shows a Java code editor interface. The top bar includes icons for file operations, a key icon, 'Java beta' (with a small 'beta' badge), a 'Run' button, and share/copy options. The main window displays a file named 'Goodbye.java'. The code is as follows:

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

A yellow arrow points to the `System.out.print("Top");` line. To the right of the code editor, there's a sidebar with the text "Powered by trinket" and "TopBottom" followed by a green horizontal bar.

# 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

The screenshot shows a Java code editor interface. At the top, there's a toolbar with icons for file operations, a key icon, and a "Java beta" tab. Below the toolbar, the code editor window displays a file named "Goodbye.java". The code contains the following Java code:

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

A large green arrow points upwards from the bottom of the slide towards the "\n" character in the third line of the code. To the right of the code editor, there's a sidebar with the text "Powered by trinket" and links for "Top" and "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:

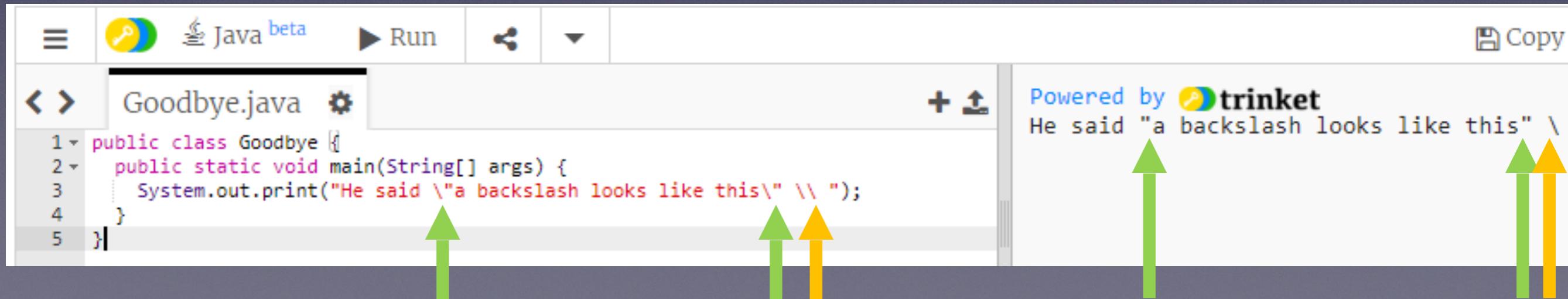
- Toolbar:** Includes icons for file operations, a key icon, "Java beta", "Run", and other development tools.
- File List:** Shows "Goodbye.java" as the active file.
- Code Editor:** Displays the following Java code:

```
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 of the program: "x→x→x→x".
- Powered by:** A "trinket" logo with 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 first two double quotes in the printed string to the corresponding double quotes in the code. A yellow arrow points from the backslash character in the printed string to the backslash character in the code.

# 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

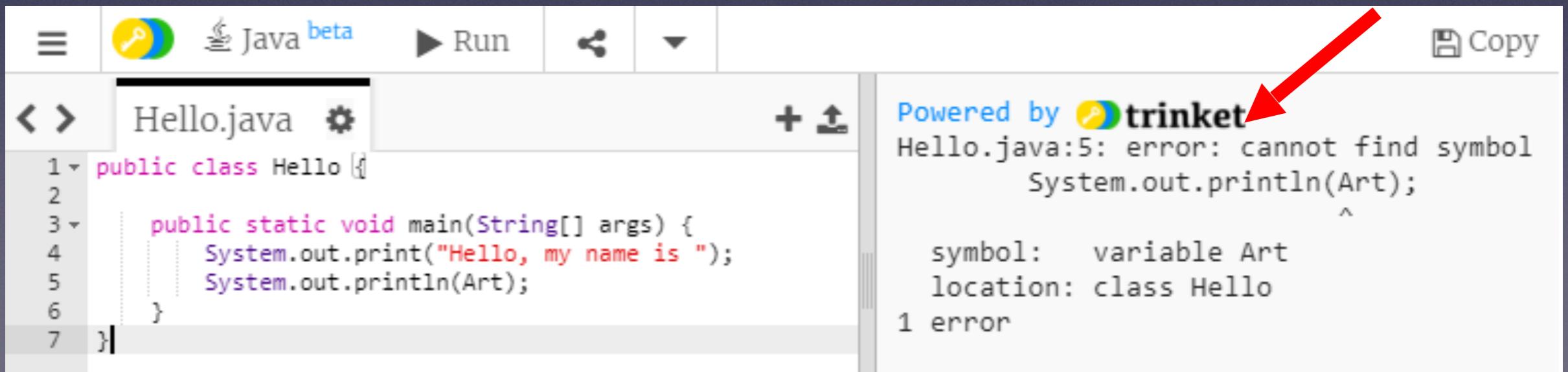
```
public class Goodbye {  
    public static void main(String[] args) {  
        System.out.print("Goodbye, ");  
        System.out.println("cruel world");  
    }  
}
```

# 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



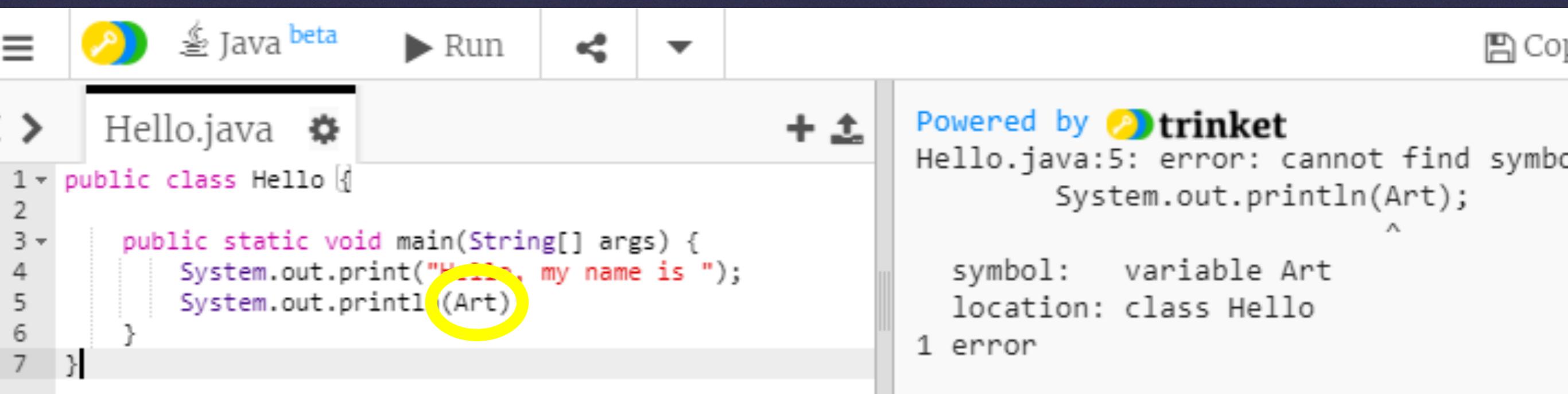
A screenshot of a Java code editor interface. The top bar includes icons for file operations, a key icon, "Java beta", a run button, and a copy button. The main area shows a file named "Hello.java" with the following code:

```
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 code contains a syntax error at line 7, where "Art" is used instead of "args". The status bar at the bottom right displays the error message: "Powered by trinket" followed by "Hello.java:5: error: cannot find symbol System.out.println(Art); symbol: variable Art location: class Hello 1 error". A red arrow points from the text "Powered by trinket" towards the error message.

# 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 beta logo, a run button, and a copy button. The left sidebar lists files: Hello.java (selected), and a plus sign icon. The main editor area contains 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 last line of the code. The status bar on the right displays the error message: "Powered by trinket", "Hello.java:5: error: cannot find symbol", "System.out.println(Art);", "symbol: variable Art", "location: class Hello", and "1 error".

# Logic error

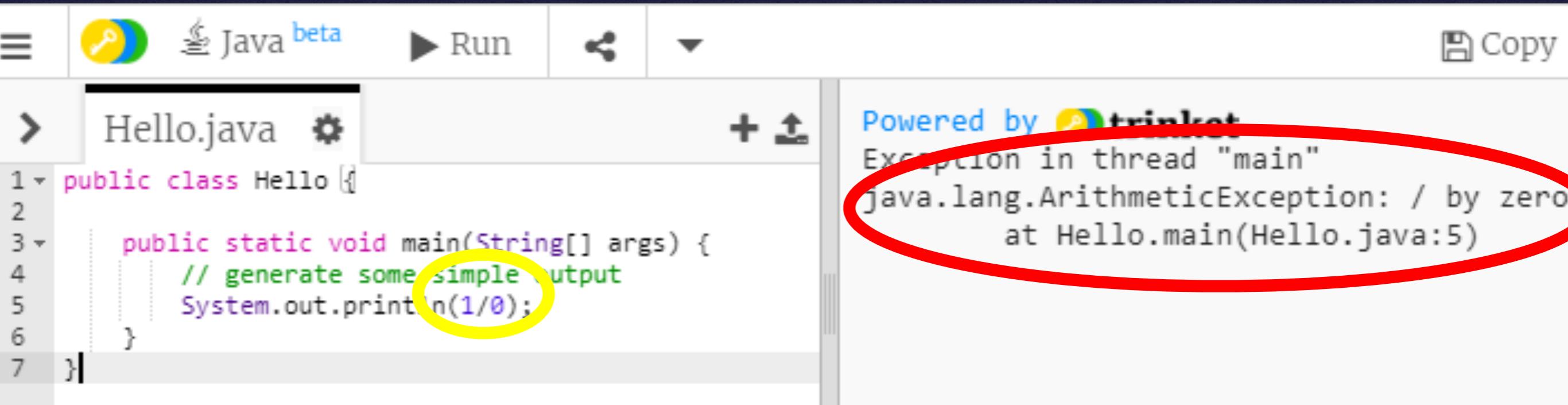
- This time I misspelled my name
- The computer doesn't know my name, so the program runs incorrectly without an error message



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 simple "Hello, world!" program with a misspelling:1 public class Hello {  
2  
3 public static void main(String[] args) {  
4 System.out.print("Hello, my name is ");  
5 System.out.println("Arg")  
6 }  
7 }The word "Arg" in the println statement is circled in yellow. The output window on the right shows the program's execution result: "Powered by trinket" followed by "Hello, my name is Arg", where "Arg" is circled in green.

# (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



```
Java beta
```

Run

Copy

Hello.java

```
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 }
```

Powered by trinket

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

# Arithmetc 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 details:

- Toolbar:** Includes icons for file operations, a key icon, Java beta logo, Run button, and share/icon buttons.
- Project Explorer:** 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 }
```

The line `System.out.println("4/4");` is circled in red.
- Output Window:** Shows the output "Powered 4/4" with the "4/4" part circled in red.
- Status Bar:** Shows the number "1".

# Literals vs. Expressions

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

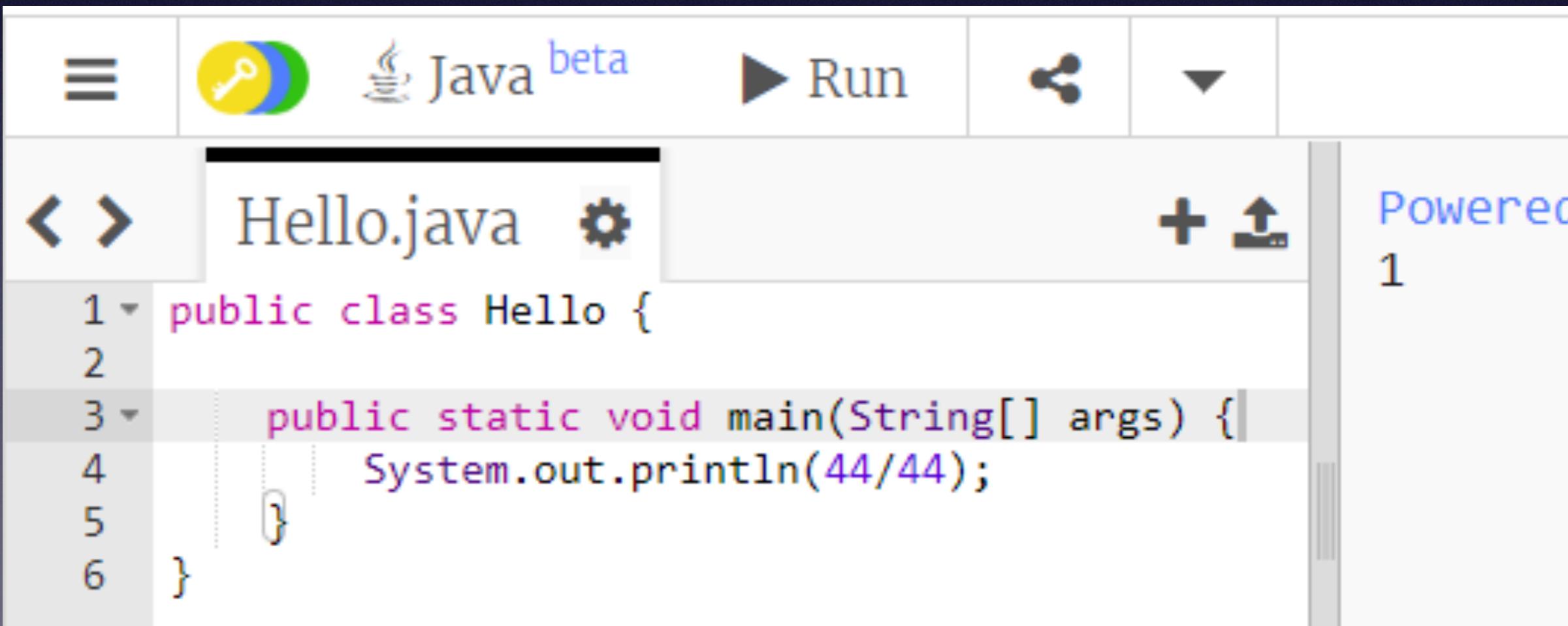
```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("4/4");  
        System.out.println(4/4);  
    }  
}
```

# 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



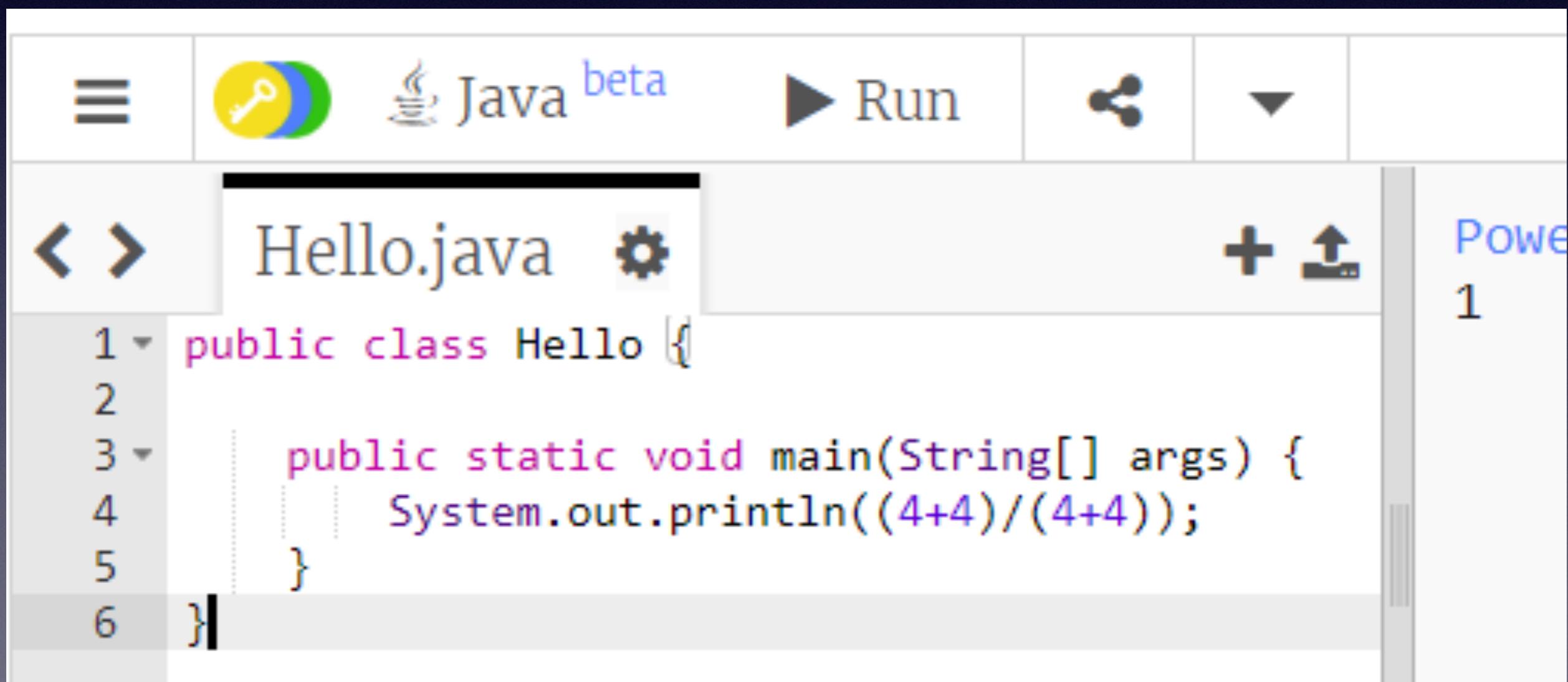
The screenshot shows a Java code editor interface. The title bar says "Java beta". The left sidebar shows a file named "Hello.java". The code editor window contains the following Java code:

```
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 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 in Java. This is likely the "bug" mentioned in the slide title.

# 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**

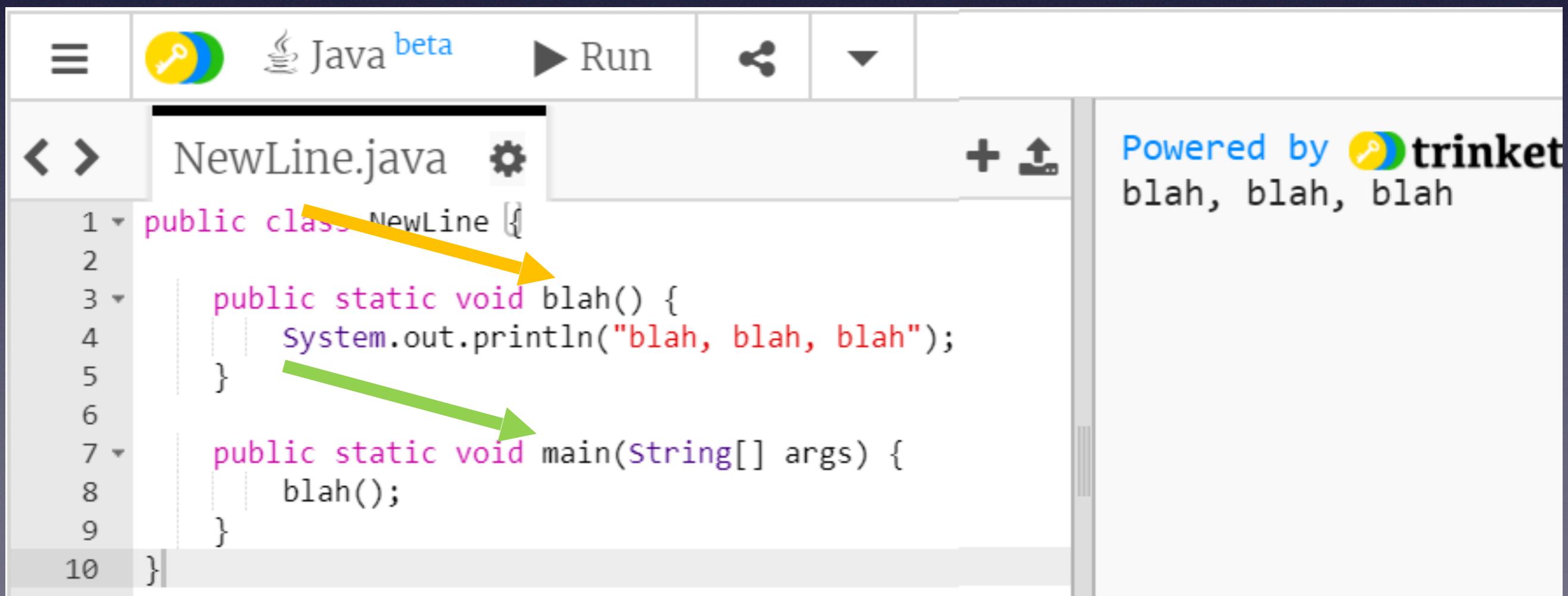
The screenshot shows a Java code editor interface. The top bar includes a menu icon, a logo with a key and gear, the text "Java beta", a "Run" button, and other icons. The left sidebar shows file navigation with "NewLine.java" selected. The main code area displays:

```
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 }
```

To the right of the code, there is a "Powered by trinket" watermark with 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();
    }
}
```

The screenshot shows a Java code editor interface. The file is named "NewLine.java". The code defines a class "NewLine" with two methods: "blah()" and "main(String[] args)". A yellow arrow points from the "void" keyword in the "blah()" method signature to the "void" keyword in the "main(String[] args)" method signature. The code editor has a toolbar with icons for file operations, a "Run" button, and a "Share" button. The status bar at the bottom right says "Powered by trinket".

# void functions (aka methods)

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

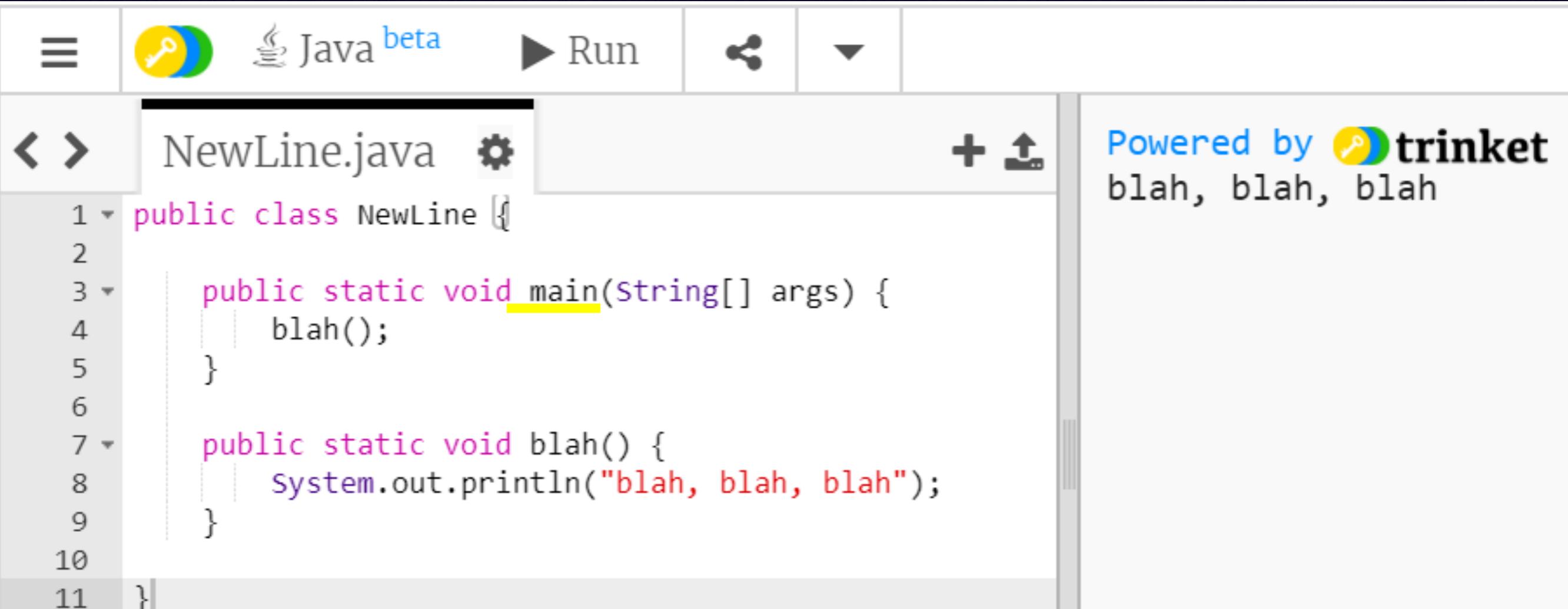
The screenshot shows a Java code editor interface. The top bar includes a menu icon, a logo with a key and gear, the text "Java beta", a "Run" button, and other icons. The left sidebar shows file navigation with "NewLine.java" selected. The main code area displays:

```
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 }
```

A yellow highlight is placed under the word "blah" in the line "blah();". To the right of the code, there is a message: "Powered by trinket" followed by the output "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 the active file, indicated by a gear icon; other listed files include "HelloWorld.java", "Fibonacci.java", and "PrimeNumbers.java". The main workspace displays the following Java code:

```
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 }
```

To the right of the code, there is a preview window showing the output: "blah, blah, blah". Below this preview, the text "Powered by trinket" is visible.

# 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**

```
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.
- Left Sidebar:** Shows navigation arrows, a file named "Main.java", and a settings gear icon.
- Code Editor:** Displays the following Java code:

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

Two green arrows point downwards from the word "num" in the function call to the variable "n" in the main method, and another green arrow points downwards from the word "word" to the variable "w". Two yellow arrows point upwards from the variable "n" in the function call to the word "num" in the function definition, and another yellow arrow points upwards from the variable "w" to the word "word".
- Right Panel:** Shows the output of the compilation process:

Powered by trinket

```
Main.java:9: error:  
incompatible types: String  
cannot be converted to int  
someFunction(w,n);  
^
```

Note: Some messages have been simplified; recompile with -Xdiags:verbose to get full output

1 error

# 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);
    }
}
```

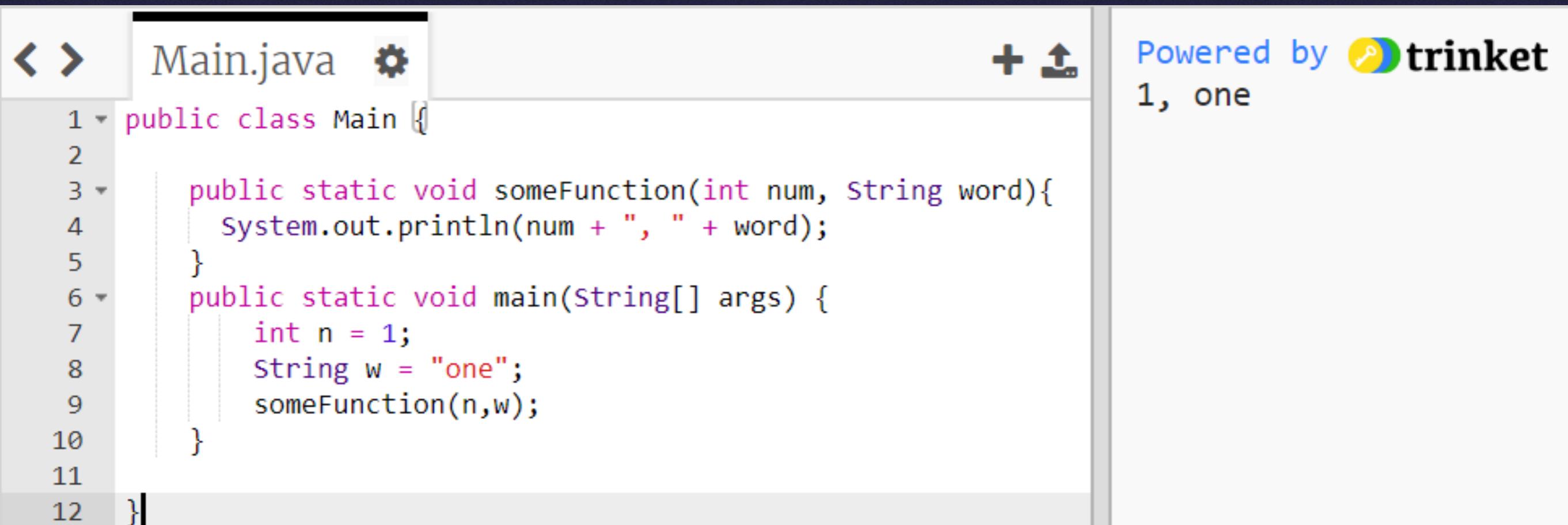
Powered by trinket

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



The screenshot shows a Java code editor with a file named `Main.java`. The code contains a `public class Main` block. Inside it, there is a `public static void someFunction(int num, String word){}` method and a `public static void main(String[] args) { ... }` method. In the `main` method, there is a line `someFunction(n,w);`. To the right of the code editor, there is a preview window showing the output of the program: `1, one`. The preview is powered by trinket.

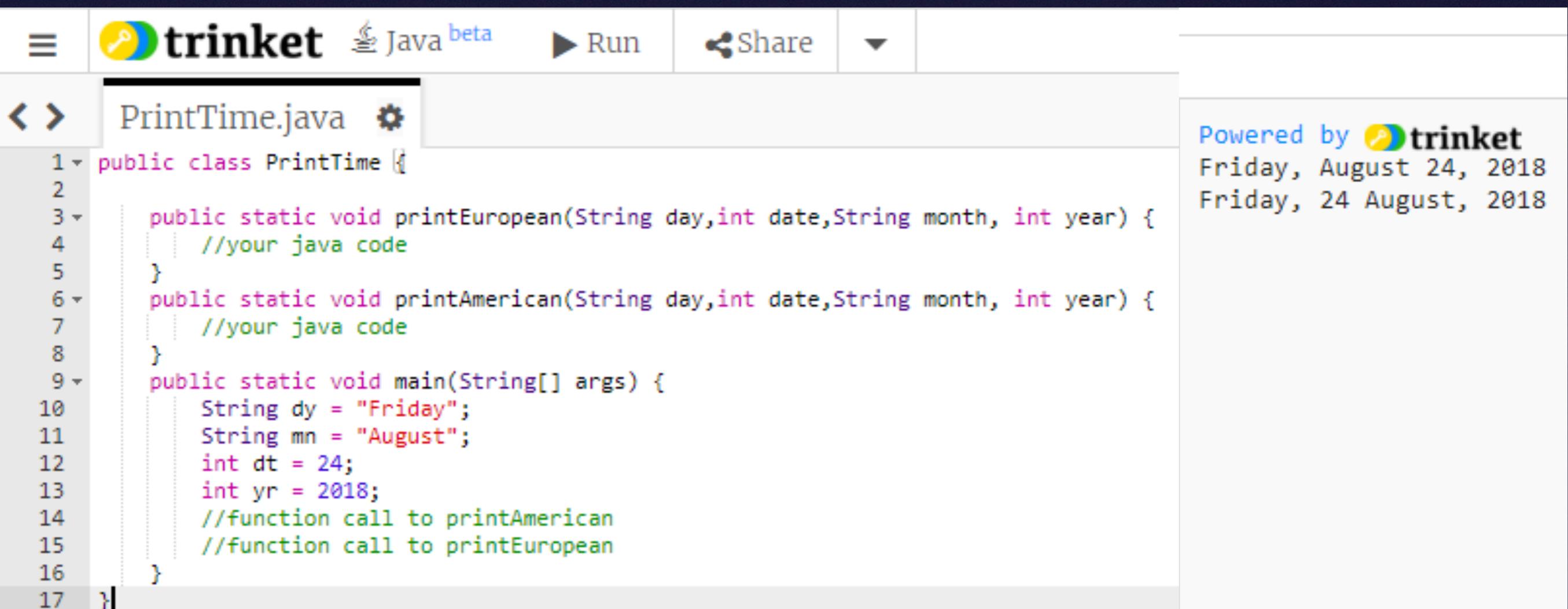
```
1 public class Main {
2
3     public static void someFunction(int num, String word){
4         System.out.println(num + ", " + word);
5     }
6     public static void main(String[] args) {
7         int n = 1;
8         String w = "one";
9         someFunction(n,w);
10    }
11
12 }
```

Powered by  trinket  
1, one

# PrintTime Function Assignment

[Go to Unit 1 Exercises](#)

Complete the PrintTime Function Assignment  
from the Functions and Parameters section

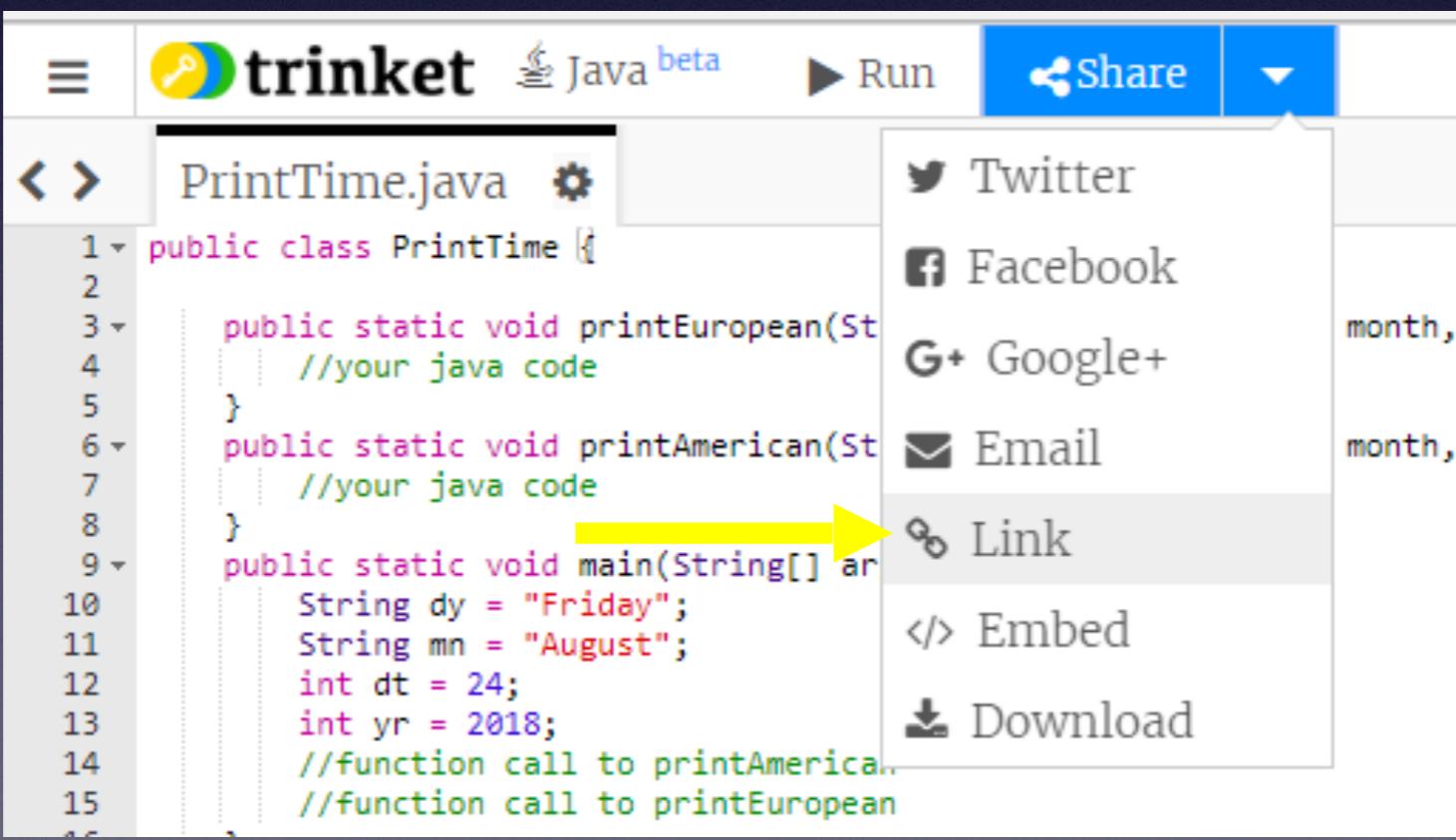


The screenshot shows the Trinket Java beta interface. The top navigation bar includes a menu icon, the Trinket logo, a Java beta badge, a Run button, a Share button, and a dropdown menu. Below the header, the code editor displays a Java file named PrintTime.java. The code contains a class definition with three methods: printEuropean, printAmerican, and main. The main method initializes variables for day, month, date, and year, and then calls the two print methods. To the right of the code editor is a results panel showing the output of the code execution.

```
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 }
```

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



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 grey.

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

# 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 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



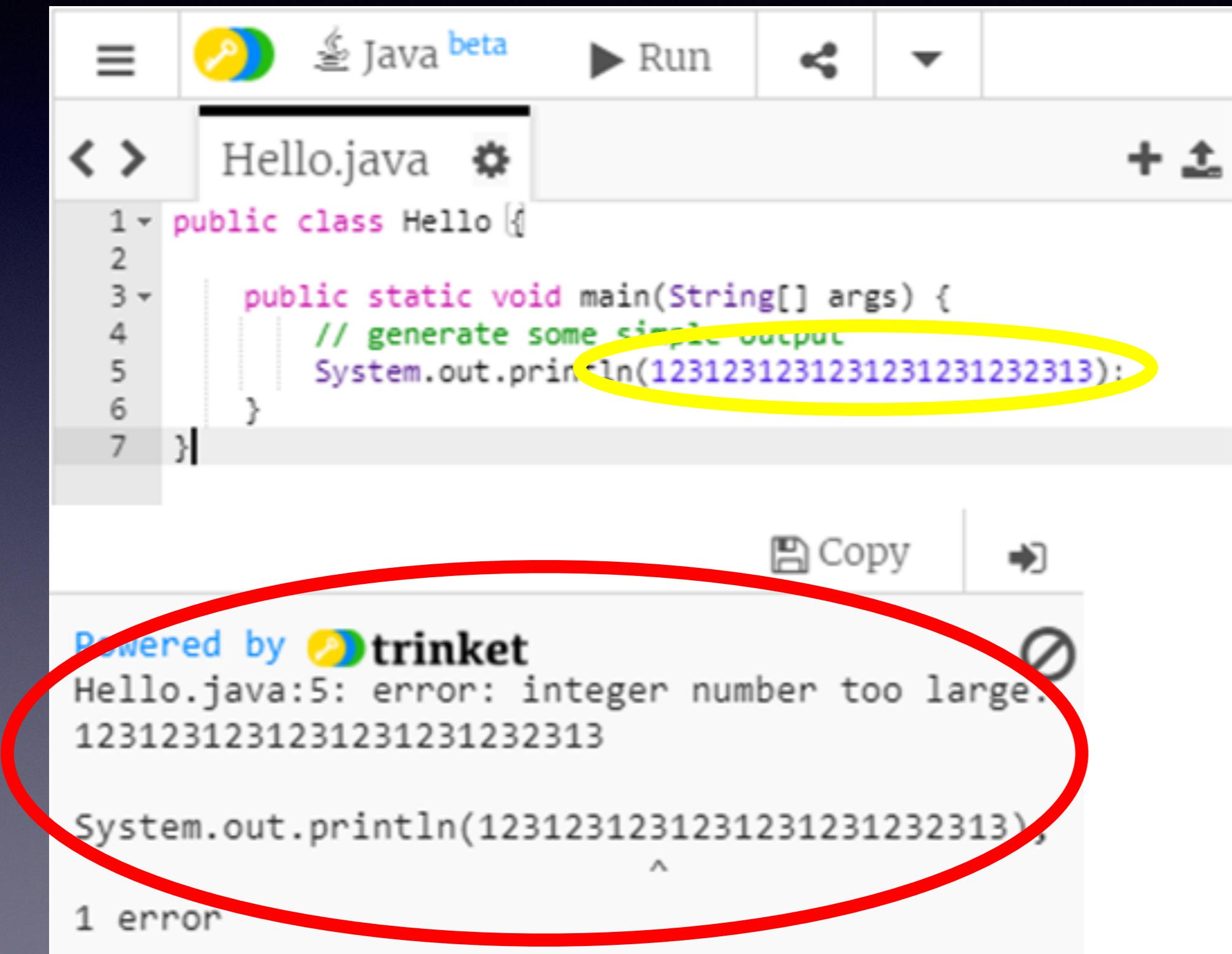
# Variables: Parking space analogy

- 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:

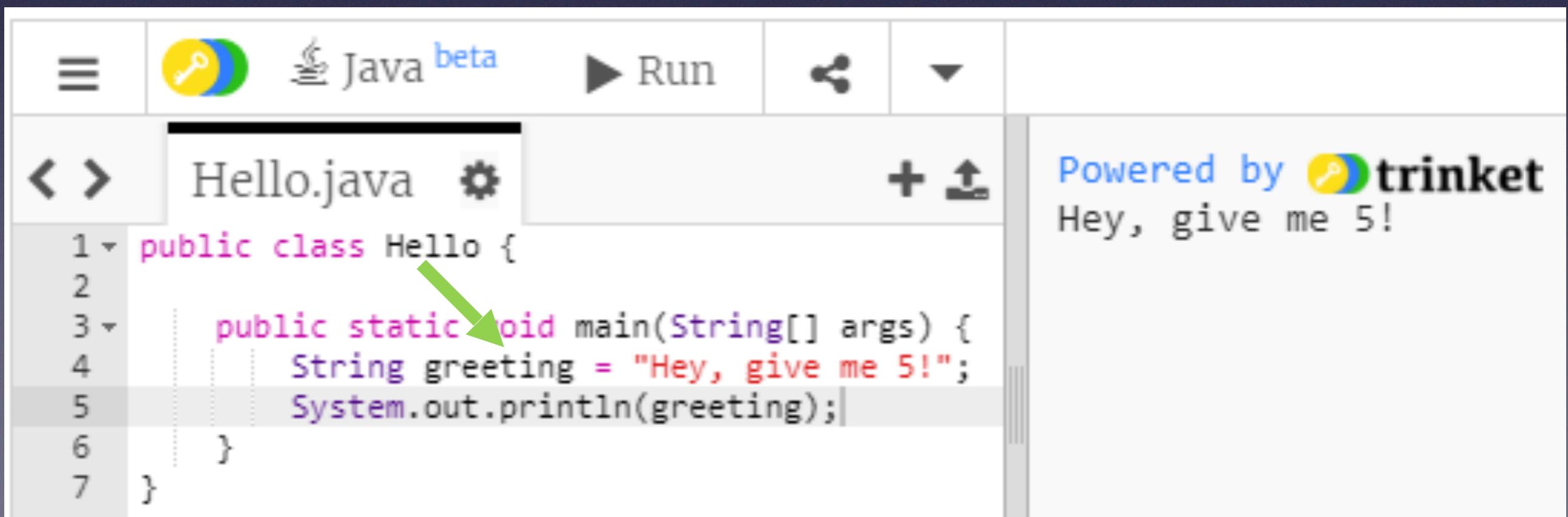
- Toolbar:** Includes icons for file operations, a key icon, "Java beta", a "Run" button, and a share icon.
- Code Area:** A file named "Hello.java" is open. The code prints the value of  $5/9.0$  using both the `double` type and the `float` type.

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

- Output Area:** Shows the output of the program. It displays two lines of text: "0.5555555555555556" and "0.555556".
- Annotations:** Two arrows point from the output lines to the corresponding printed values in the code area. A yellow arrow points to the first output line, and a green arrow points to the second.
- Powered by:** The text "Powered by trinket" is visible on the right side of the interface.

# 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 ("")



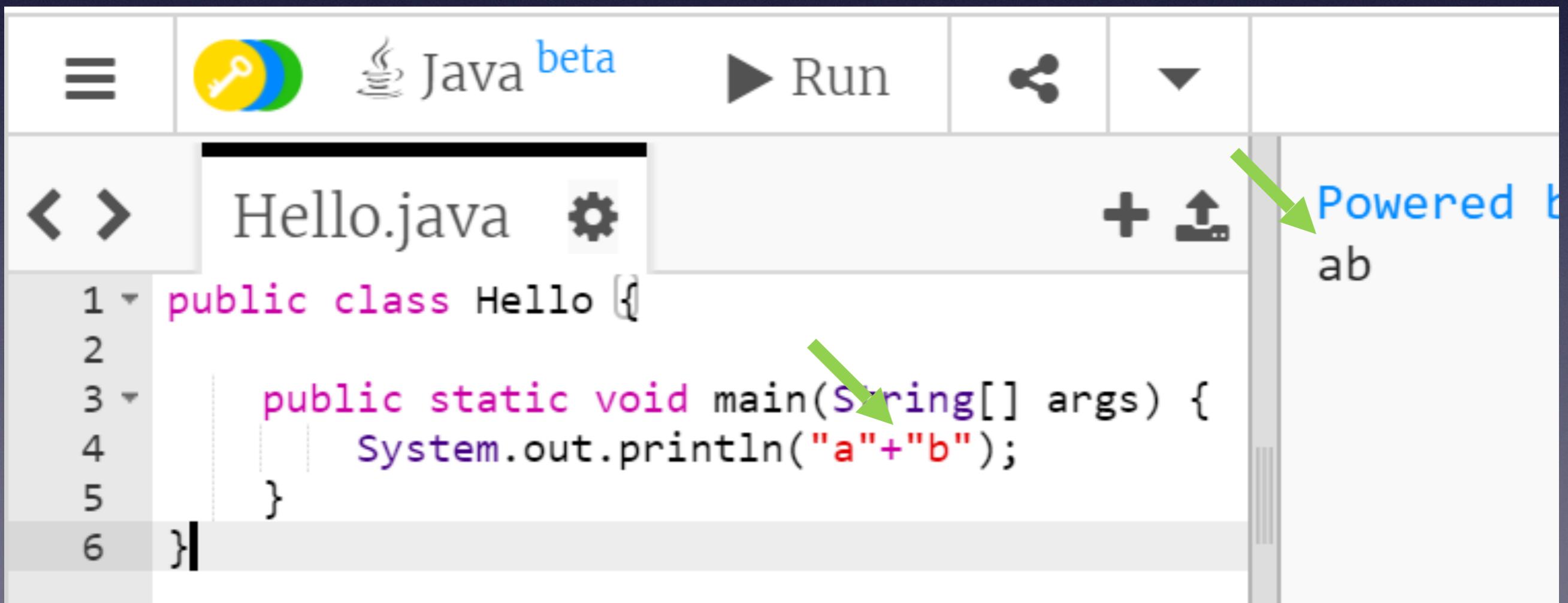
A screenshot of a Java code editor interface. The top bar shows the Java logo and the word "Java beta". Below the bar, a file named "Hello.java" is open. The code editor displays the following Java code:

```
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 editor, there is a preview window showing the output of the program: "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  $\text{"3Hello"}$
- $\text{"Hello"} + 1$  is  $\text{"Hello1"}$ , and  $\text{"Hello1"} + 2$  is  $\text{"Hello12"}$

```
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

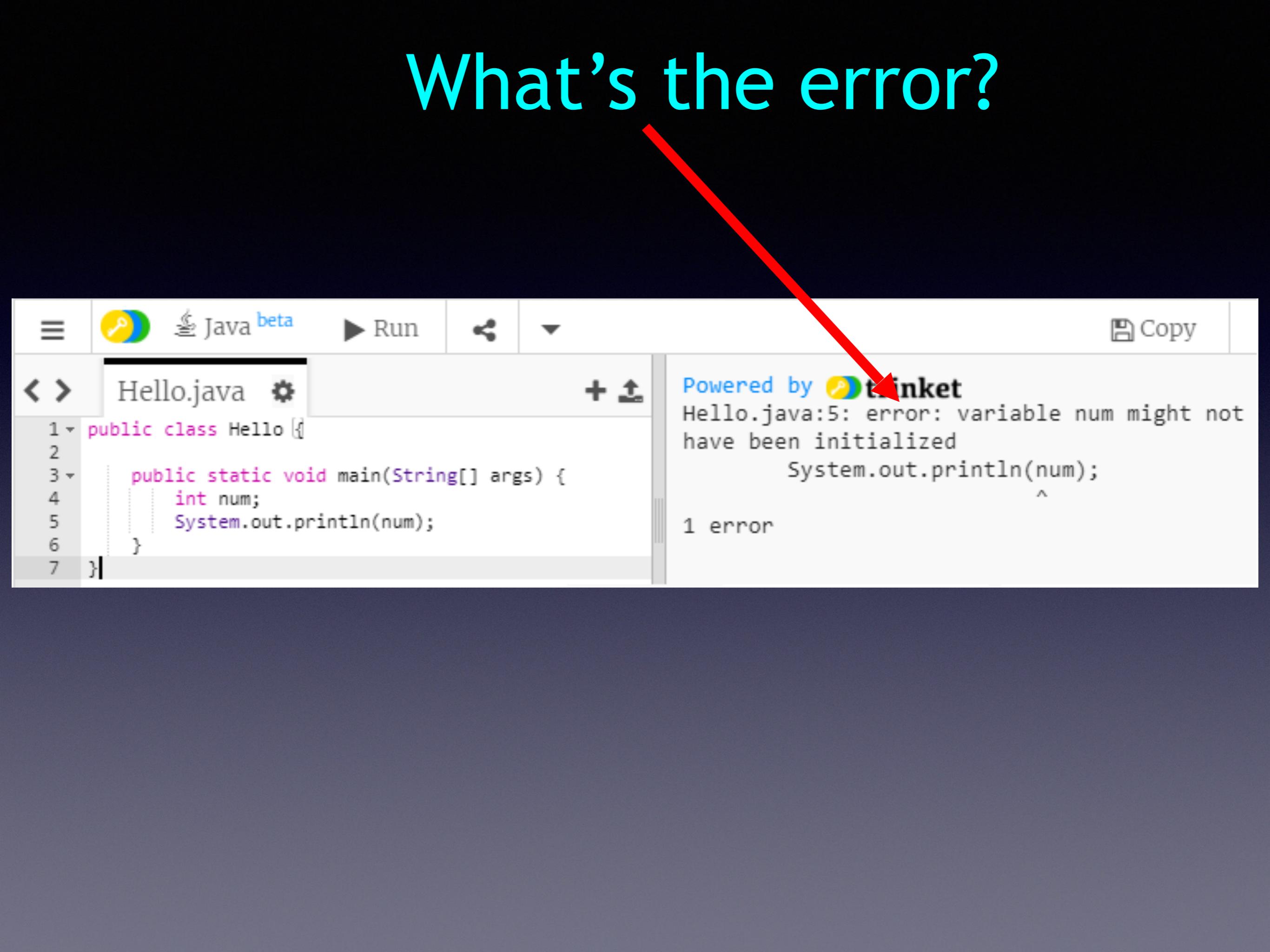
# Variable names and keywords

- 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, a 'Java beta' tab, a 'Run' button, and a 'Copy' button. The main area shows a file named 'Hello.java' with the following code:

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

The output panel on the right displays the error message: "Powered by  tinket". Below it, the error details are shown: "Hello.java:5: error: variable num might not have been initialized" followed by the line "System.out.println(num);". An arrow points from the question mark in the title to the 'tinket' logo in the error message.

Powered by  tinket

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

1 error

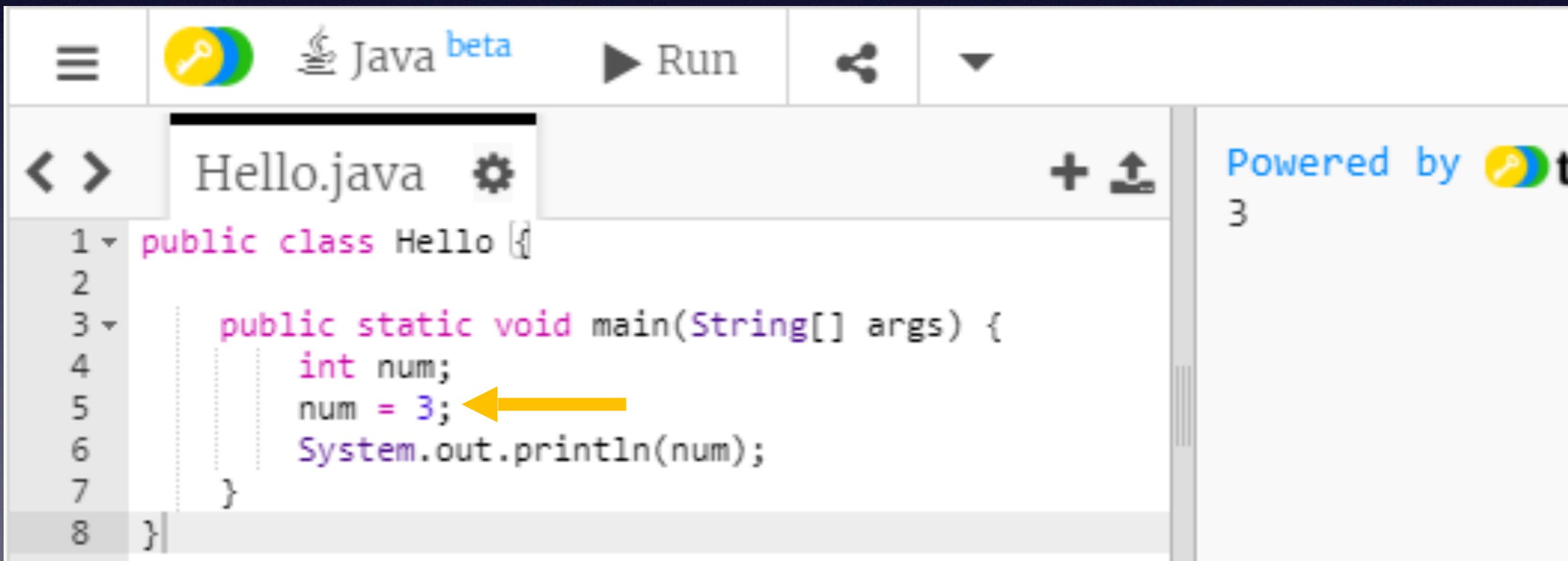
# Variable initialization

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

The screenshot shows a Java code editor interface. The file being edited is named "Hello.java". The code contains a simple main method:public class Hello {  
 public static void main(String[] args) {  
 int num;  
 System.out.println(num);  
 }  
}A syntax error is highlighted at the line "System.out.println(num);". The error message on the right side of the editor states: "Powered by trinket" and "Hello.java:5: error: variable num might not have been initialized System.out.println(num); ^". Below the error message, it says "1 error".

# Variable initialization

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



A screenshot of a Java code editor interface. The top bar includes a key icon, the text "Java beta", a "Run" button, and a share icon. The main area shows a file named "Hello.java". The code is:

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

The line "num = 3;" is highlighted with a yellow arrow pointing to the assignment operator "=".

# 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?

$$5 \sqrt{8}$$

# Modulus and integer division

Remember how you did math in grade school?

$$\begin{array}{r} 1 \\ 5 \sqrt{8} \end{array}$$

# Modulus and integer division

Remember how you did math in grade school?

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

# Modulus and integer division

Remember how you did math in grade school?

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

# Modulus and integer division

The modulus operator gives the remainder of an integer division expression

$$\text{8/5}$$

$$\text{8\%5}$$

$$5 \overline{)8} \quad \begin{array}{r} 1 \\ -5 \\ \hline 3 \end{array}$$

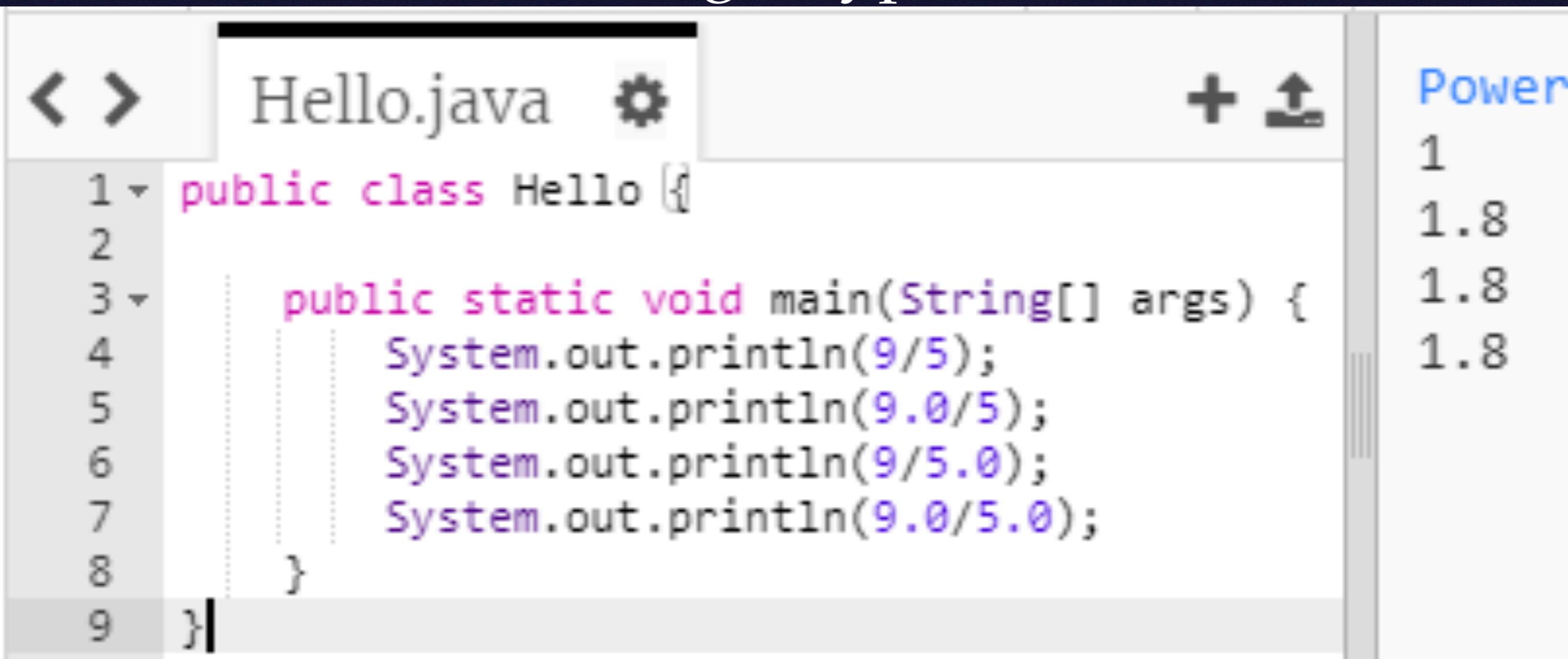
# Kahoot!

**Integer Division in Java**

$$\begin{array}{r} 0 \\ 8 \overline{)5} \\ \underline{-0} \\ 5 \end{array}$$

# 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



```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println(9/5);  
        System.out.println(9.0/5);  
        System.out.println(9/5.0);  
        System.out.println(9.0/5.0);  
    }  
}
```

The screenshot shows a Java code editor with a file named "Hello.java". The code contains four `System.out.println` statements. The first two statements involve integer division: `9/5` and `9.0/5`. The last two statements involve mixed-type division: `9/5.0` and `9.0/5.0`. To the right of the code editor, there is a vertical sidebar titled "Power" with the following values listed:  
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)

The screenshot shows a Java code editor with a file named "Hello.java". The code defines a class "Hello" with a main method that initializes variables for hour, minute, and second, and then prints three calculated values: time since midnight, time remaining today, and the percentage of the day passed.

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

The output window on the right is titled "Powered by trinket" and displays the results of the program execution:

- 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`

The screenshot shows a code editor window titled "Echo.java". The code is as follows:

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

To the right of the code, there is an output panel. It says "Powered by trinket". Below that, it shows the prompt "Type an integer: 42" and the response "You typed: 42". A yellow arrow points to the first line of the code, another green arrow points to the line where the scanner is created, and a third yellow arrow points to the line where the integer is read.

# The types must match

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

The screenshot shows a code editor window titled "Echo.java". The code is as follows:

```
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 }
```

A green arrow points to the line `String someInteger = in.nextInt();`. Another green arrow points from the word `nextInt()` back to the line `String someInteger = in.nextInt();`. To the right of the code editor, the output is displayed:

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/](https://www.tools4noobs.com/online_tools/seconds_to_hh_mm_ss/)

online tools/

seconds to hh mm ss/

- 10,000 seconds is 2 hours 46 minutes and 10 seconds

The screenshot shows a web browser window with the URL [https://www.tools4noobs.com/online\\_tools/seconds\\_to\\_hh\\_mm\\_ss/](https://www.tools4noobs.com/online_tools/seconds_to_hh_mm_ss/) in the address bar. The page title is "Convert seconds to HH:MM:SS". The main content area includes a navigation bar with links like Home, Online tools, Convert seconds to HH:MM:SS, and a link to another tool. Below the navigation is a form with a "Seconds" input field containing "10000" and a "Convert" button. At the bottom, the result is displayed as "Result: 02:46:40."

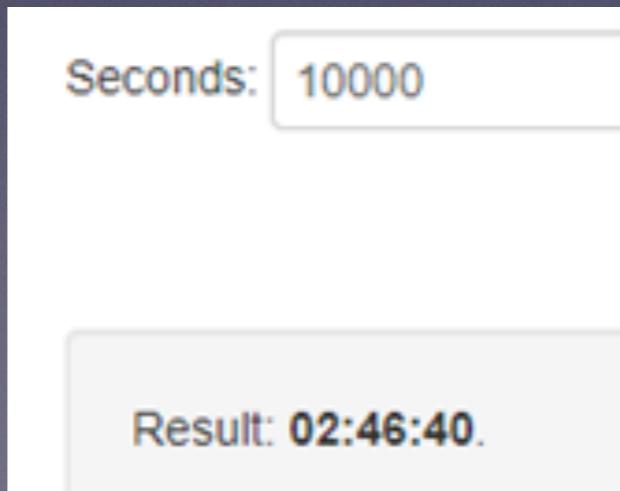
## Using modulus

$10,000 / (60 * 60)$  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

The screenshot shows a code editor window titled "Echo.java" with Java code. The code imports java.util.Scanner, defines a public class Echo with a main method, and calculates the equivalent time in hours, minutes, and seconds from a given number of seconds. The editor has line numbers on the left and syntax highlighting. To the right, the output is shown, powered by trinket. It prompts for a number of seconds (10000), then displays the result: 10000 seconds is: 2 hours 46 minutes and 40 seconds.

```
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 `!`

<code>&amp;&amp;</code>	T	F	<code>  </code>	T	F	!	T	F
T	T	F	T	T	T	F	T	T
F	F	F	F	T	F		F	T

# Logical Operators

- What would be printed?

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

$\&\&$	T	F	$\ $	T	F	!	T	F
T	T	F	T	T	T	F	F	T
F	F	F	F	T	F			

# Logical Operators

■ What would be printed?

```
int nCount = 150;
```

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

&&	T	F		T	F	!	T	F
T	T	F	T	T	T	F	T	
F	F	F	F	T	F			

The screenshot shows a Java code editor with the file "Hello.java" open. The code defines a class "Hello" with a main method that prints a logical expression to the console. The output on the right indicates the result is "false".

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

Powered by trinket  
false

# Logical Operators

- Now what would be printed?

```
int nCount = 50;
```

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

&&	T	F		T	F	!	T	F
T	T	F	T	T	T	F	F	T
F	F	F	F	T	F			

# Logical Operators

$\&\&$	T	F	$\ $	T	F	!	T	F
T	T	F	T	T	T	F	F	T
F	F	F	F	T	F			

- What would be printed?

```
int nCount = 50;
```

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

The screenshot shows a code editor interface for Java. The file is named "Hello.java". The code contains a single method main with a variable nCount set to 50. The output window on the right shows the result of the println statement: "true".

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

Powered by trinket  
true

# How would Java evaluate these expressions?

$\&\&$	T	F	$\ $	T	F	!	T	F
T	T	F	T	T	T	F	T	F
F	F	F	F	T	F			

```
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

$\&\&$	T	F
T	T	F
F	F	F

$!$	T	F
T	F	T

- **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

<code>&amp;&amp;</code>	T	F	<code>  </code>	T	F	!	T	F
T	T	F	T	T	T	F	F	T
F	F	F	F	T	F			

- 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

<code>&amp;&amp;</code>	T	F	<code>  </code>	T	F	!	T	F
T	T	F	T	T	T	F	F	T
F	F	F	F	T	F			

- What does  
 $1==0 \quad \&\& \quad 274/17 > 784/461$   
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

$!(A \And B)$  is the same as  $\neg A \Or \neg B$

$!(A \Or B)$  is the same as  $\neg A \And \neg B$

# 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)**

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 a "Copy" button.
- File List:** Shows "NewLine.java" as the active file.
- Code Editor:** Displays the following Java code:

```
1 public class NewLine {  
2  
3     public static void main(String[] args) {  
4         if(1/0 != 0){  
5             System.out.println("Hello World!");  
6         }  
7     }  
8 }
```

A blue arrow points to the line `if(1/0 != 0){`.
- Output Area:** Shows the error message:

```
Powered by trinket  
exception in thread "main"  
java.lang.ArithmaticException: / by  
zero  
at  
NewLine.main(NewLine.java:4)
```

A yellow oval highlights the "Powered by trinket" text.

# 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



The screenshot shows the Trinket Java beta interface. At the top, there's a toolbar with icons for file operations, a key icon, "Java beta", and a "Run" button. Below the toolbar, a tab bar shows "NewLine.java". The code editor contains the following Java code:

```
1 public class NewLine {  
2  
3     public static void main(String[] args) {  
4         if(1==1 || 1/0 != 0){  
5             System.out.println("Hello World!");  
6         }  
7     }  
8 }
```

To the right of the code editor is the output window, which displays "Powered by trinket" and "Hello World!". A blue arrow points from the word "anything" in the list above to the logical OR operator "||" in the code, and a yellow arrow points from the word "divide" to the division operation "1/0".

# More short circuit evaluation

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

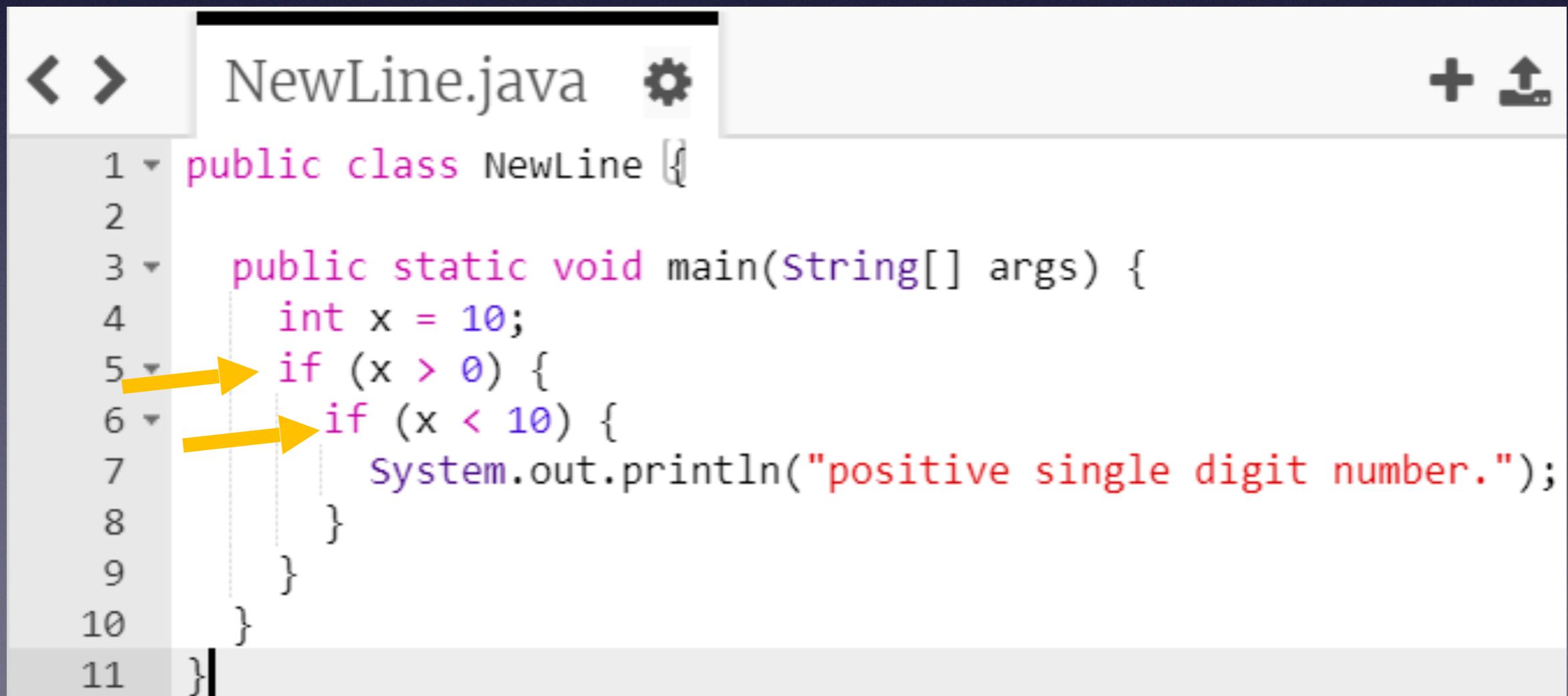
The screenshot shows a Java code editor interface. The top bar includes icons for file operations, a key icon, "Java beta", a run button, and a copy button. The code editor window displays a file named "NewLine.java". The code contains a main method that attempts to divide 1 by 0, which results in an ArithmeticException. A blue arrow points to the division operation. The output pane on the right shows the exception details: "Powered by trinket", "Exception in thread "main"" followed by the stack trace "java.lang.ArithmeticException: / by zero at NewLine.main(NewLine.java:4)".

```
1 public class NewLine {
2
3     public static void main(String[] args) {
4         if(1/0 != 0 || 1==1){
5             System.out.println("Hello World!");
6         }
7     }
8 }
```

Powered by trinket  
Exception in thread "main"  
java.lang.ArithmeticException: / by zero  
at  
NewLine.main(NewLine.java:4)

# 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



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

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

**absoluteValue()** and **main()**

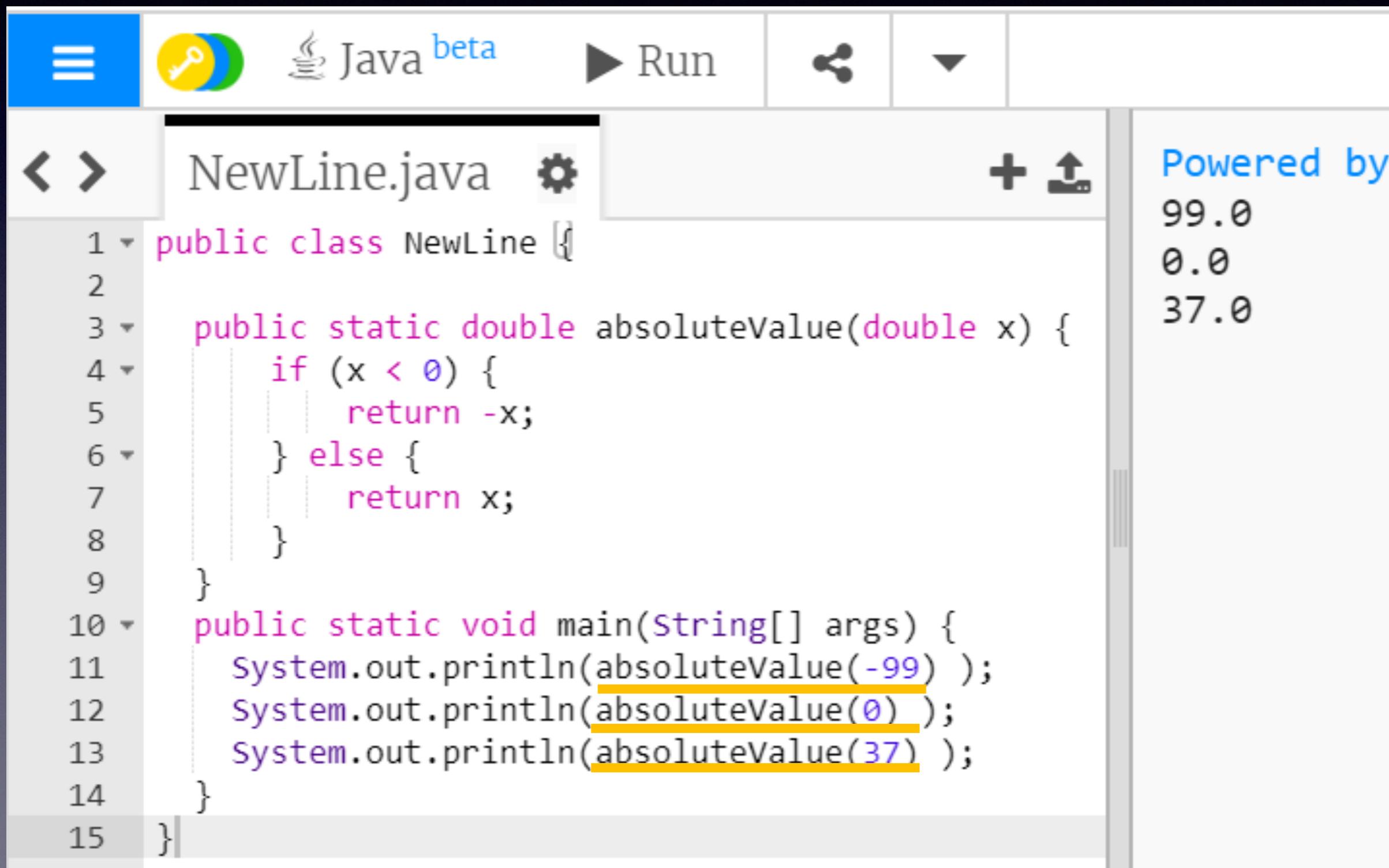
```
NewLine.java
public class NewLine {
    public static double absoluteValue(double x) {
        if (x < 0) {
            return -x;
        } else {
            return x;
        }
    }
    public static void main(String[] args) {
        System.out.println(absoluteValue(-99));
        System.out.println(absoluteValue(0));
        System.out.println(absoluteValue(37));
    }
}
```

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

- Header:** Java beta
- File:** NewLine.java
- Content:** The code defines a class named NewLine with two methods:
  - absoluteValue(double x):** A method that takes a double as input and returns its absolute value. It uses an if-else conditional statement to determine whether the input is negative or non-negative.
  - main(String[] args):** The main entry point of the program, which prints the results of calling absoluteValue() for three specific values: -99, 0, and 37.
- Annotations:** A yellow arrow highlights the start of the `absoluteValue` method definition, and a green arrow highlights the start of the `main` method definition.
- Right Panel:** Shows performance metrics: Powered by 99.0, 0.0, and 37.0.

# A program using `absoluteValue()`

- Three function calls to `absoluteValue()`



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

- Header:** Java beta
- File:** NewLine.java
- Code Content:**

```
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 }
```
- Output Panel:** Shows the results of the three println statements:
  - 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

The screenshot shows a Java code editor interface. The file is named `NewLine.java`. The code defines a class `NewLine` with a static method `absoluteValue` that returns the absolute value of a double. The method uses an if-else conditional to determine the sign of `x` and return the appropriate value. Following the method definition, there is an `System.out.println` statement on line 9. This statement is highlighted in red, indicating it is an error. A tooltip on the right side of the screen states: "Powered by trinket" and "NewLine.java:9: error: unreachable statement System.out.println("This line is dead.");". The line number 9 is also indicated in the tooltip.

```
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         System.out.println("This line is dead.");  
10    }  
11    public static void main(String[] args) {  
12    }  
13 }
```

# An `isSingleDigit()` function

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

```
public class NewLine {  
    public static boolean isSingleDigit(int x) {  
        if (x > -10 && x < 10) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
    public static void main(String[] args) {  
        System.out.println(isSingleDigit(-99));  
        System.out.println(isSingleDigit(0));  
        System.out.println(isSingleDigit(37));  
        System.out.println(isSingleDigit(7));  
    }  
}
```

Powered  
false  
true  
false  
true

# A program using `isSingleDigit()`

- Four function calls to `isSingleDigit()`

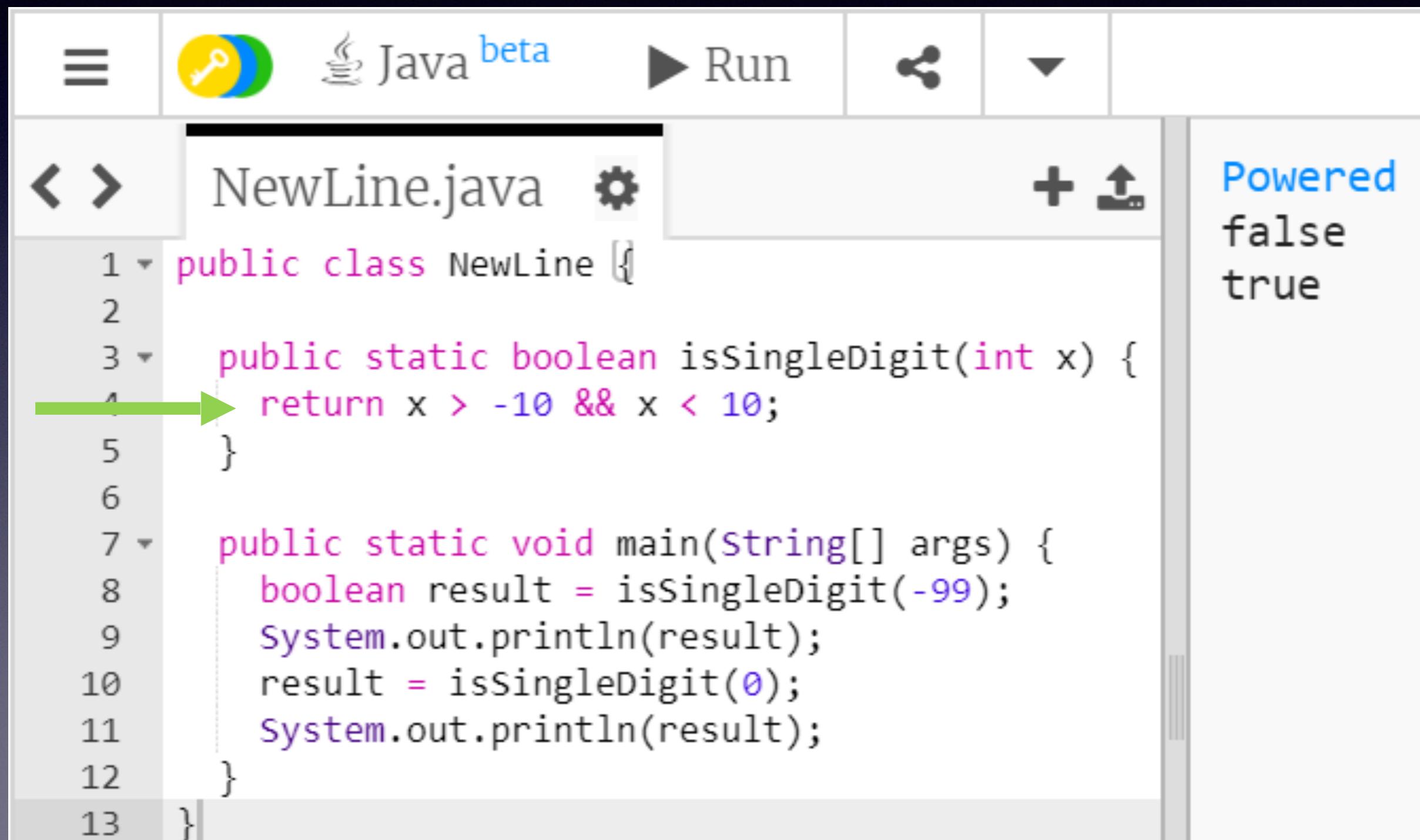
NewLine.java

```
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 `isSingleDigit()`

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



The screenshot shows a Java IDE interface with the following details:

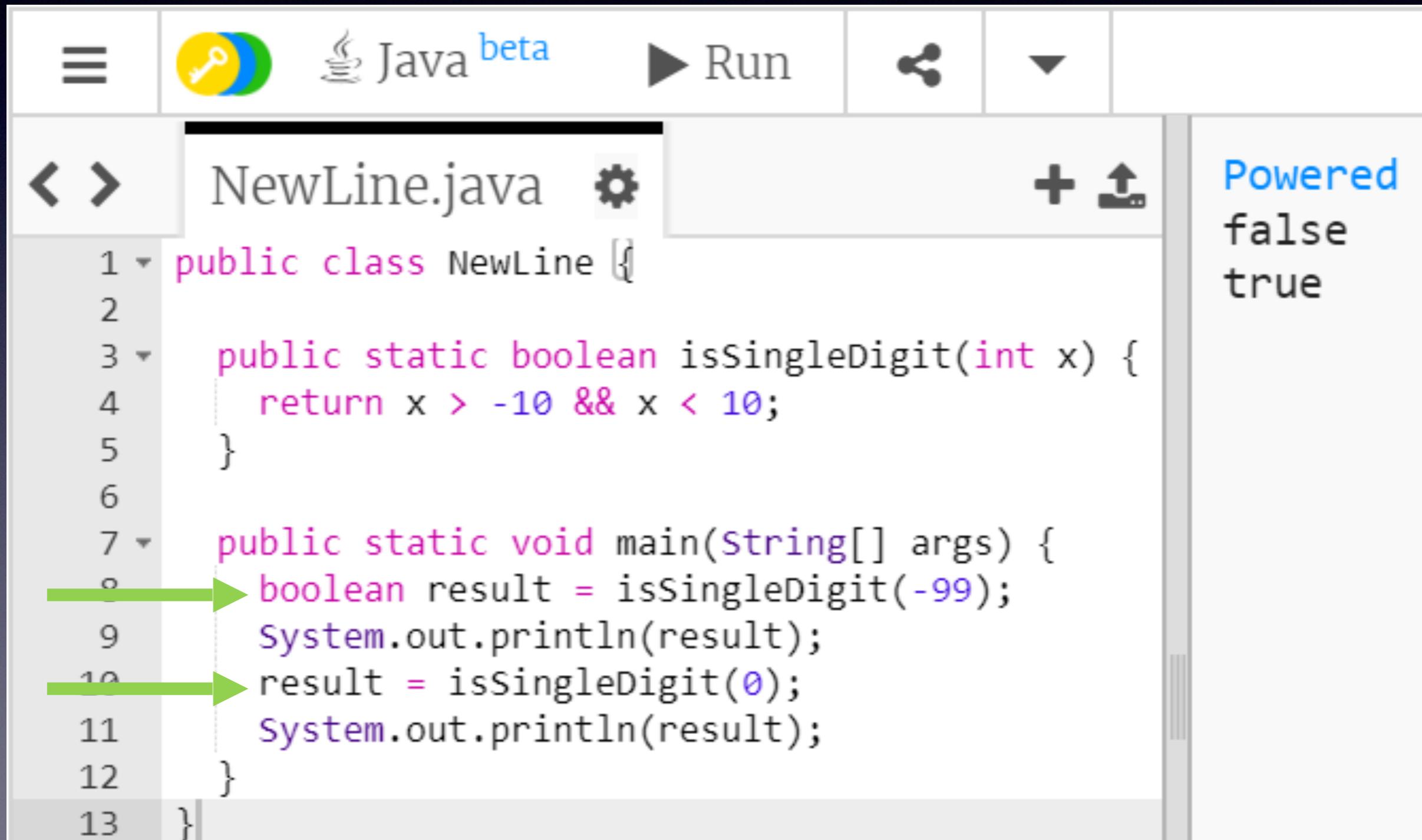
- Toolbar:** Includes icons for file operations, a key icon, "Java beta", "Run", and others.
- Project Explorer:** Shows a single file named "NewLine.java".
- Code Editor:** Displays the following Java code:

```
public class NewLine {
    public static boolean isSingleDigit(int x) {
        return x > -10 && x < 10;
    }
    public static void main(String[] args) {
        boolean result = isSingleDigit(-99);
        System.out.println(result);
        result = isSingleDigit(0);
        System.out.println(result);
    }
}
```
- Output:** On the right, the output window shows:

Powered  
false  
true

# Storing the return value

- Again, just another way of doing the same thing

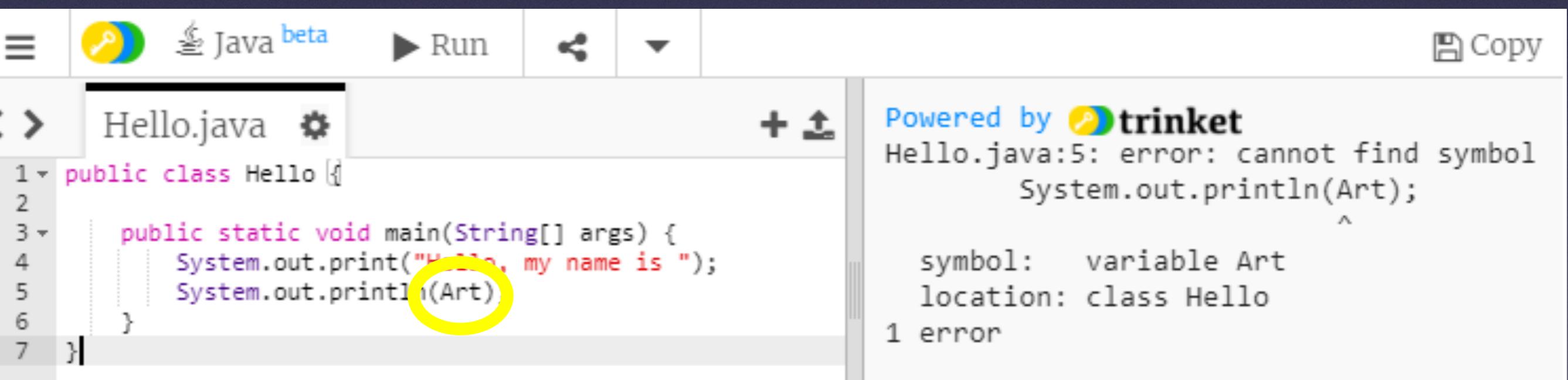


```
NewLine.java
public class NewLine {
    public static boolean isSingleDigit(int x) {
        return x > -10 && x < 10;
    }
    public static void main(String[] args) {
        boolean result = isSingleDigit(-99);
        System.out.println(result);
        result = isSingleDigit(0);
        System.out.println(result);
    }
}
```

The screenshot shows a Java development environment with the file "NewLine.java" open. The code defines a class "NewLine" with a static method "isSingleDigit" that returns true if an integer is between -10 and 10. The "main" method calls this method twice and prints the results. Two lines of code are highlighted with green arrows: line 9, where "result" is assigned the value of "isSingleDigit(-99)", and line 10, where "result" is assigned the value of "isSingleDigit(0)". The output window on the right shows the results: "false" for the first call and "true" for the second.

# 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



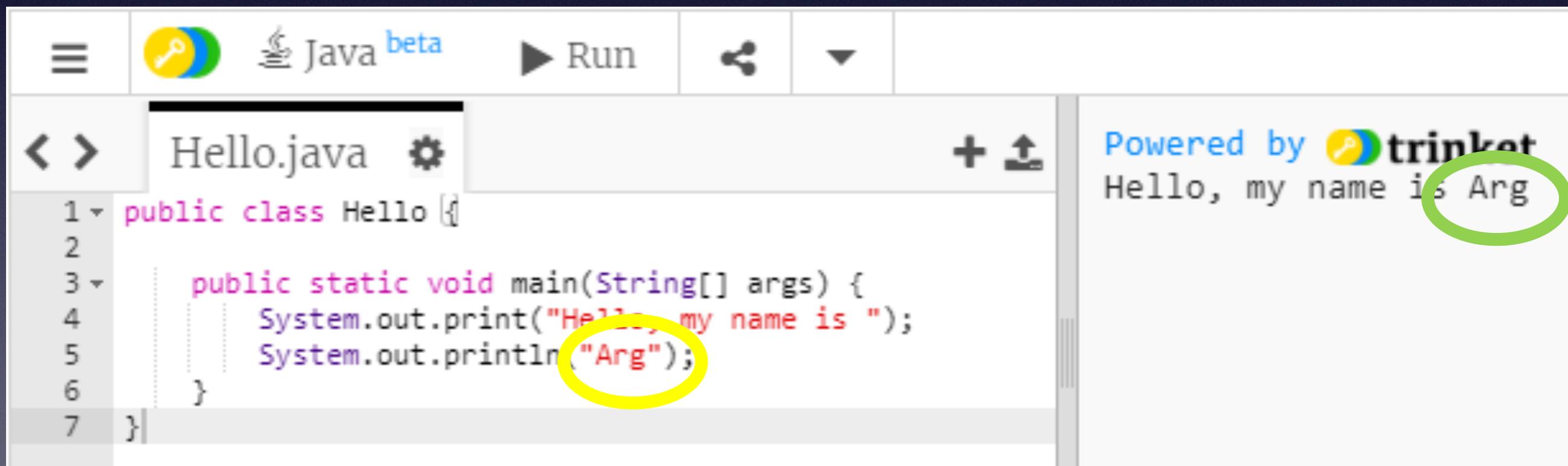
The screenshot shows a Java code editor interface. The file being edited is `Hello.java`. The code contains a `public class Hello` block with a `main` method. In the `main` method, there is a line of code `System.out.println(Art);` where the variable `Art` is misspelled. A yellow circle highlights this line. To the right of the editor, the output window displays the error message: "Powered by trinket", "Hello.java:5: error: cannot find symbol", "System.out.println(Art);", "symbol: variable Art", "location: class Hello", and "1 error".

```
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 }
```

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



The screenshot shows a Java code editor interface with a file named "Hello.java". The code contains a simple "Hello, world!" program. A yellow oval highlights the misspelling "Arg" in the line `System.out.println("Arg");`. A green oval highlights the misspelling "Arg" in the output text "Hello, my name is Arg".

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

Powered by trinket  
Hello, my name is Arg

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

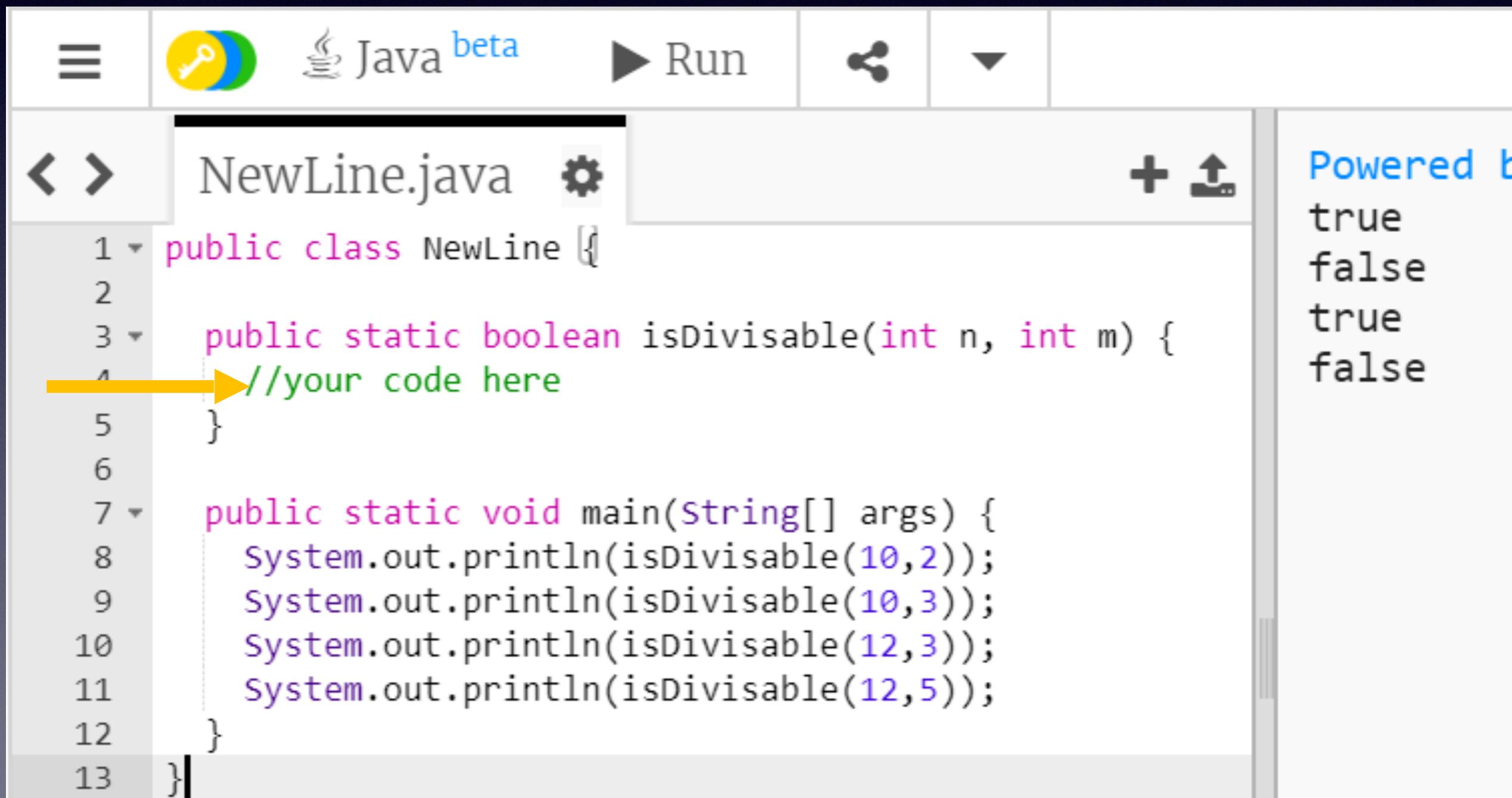
- Try experimenting



```
public class NewLine {  
    public static boolean isSingleDigit(int x) {  
        return x > -10 && x < 10;  
    }  
    public static void main(String[] args) {  
        isSingleDigit(-99);  
        isSingleDigit(0);  
    }  
}
```

# Write a `isDivisible()` function

- Hint: use modulus

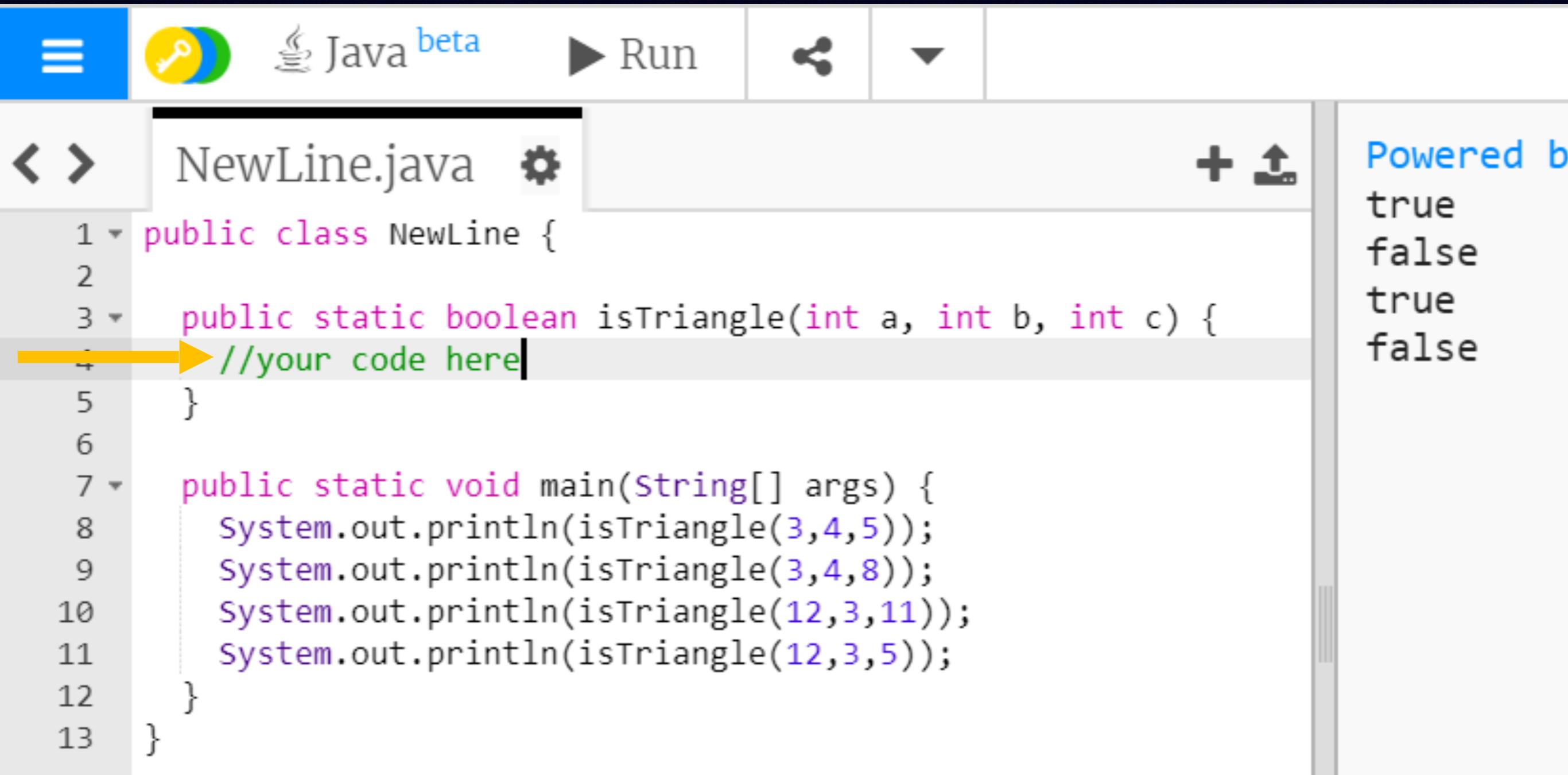


```
NewLine.java
public class NewLine {
    public static boolean isDivisible(int n, int m) {
        //your code here
    }
    public static void main(String[] args) {
        System.out.println(isDivisible(10,2));
        System.out.println(isDivisible(10,3));
        System.out.println(isDivisible(12,3));
        System.out.println(isDivisible(12,5));
    }
}
```

Powered by  
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.



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

Powered by  
true  
false  
true  
false

# Calling functions with return values

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

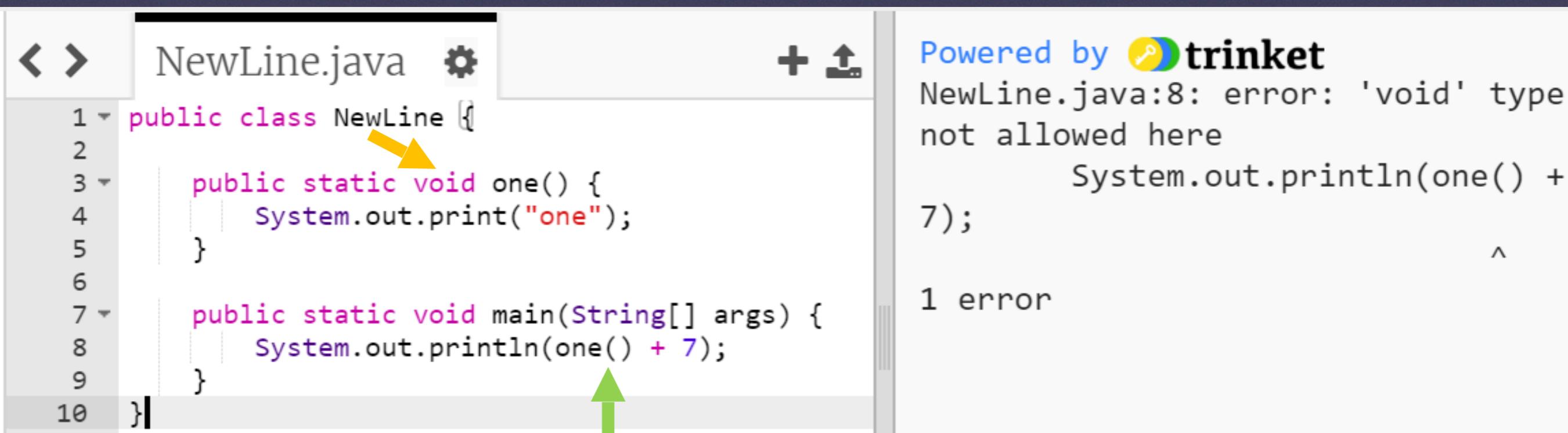


```
public class NewLine {  
    public static boolean issingleDigit(int x) {  
        return x > -10 && x < 10;  
    }  
    public static void main(String[] args) {  
        issingleDigit(-99);  
        issingleDigit(0);  
    }  
}
```

# 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



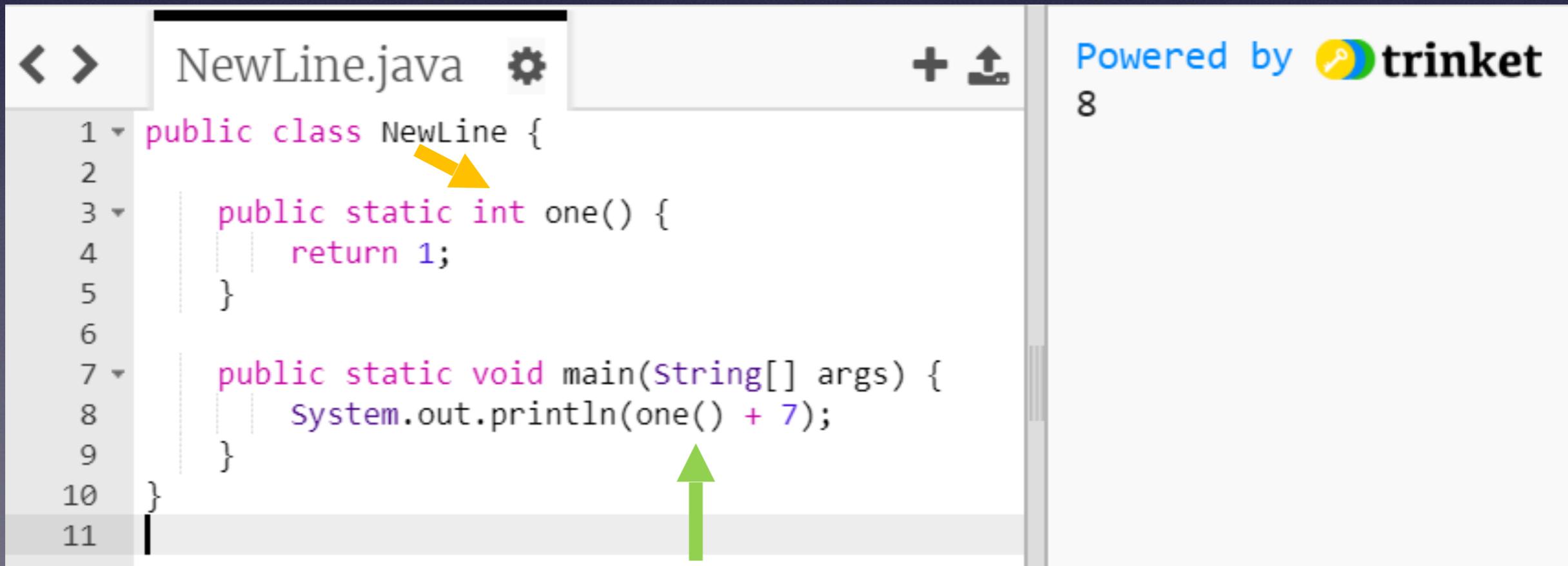
```
< > NewLine.java + 
1 public class NewLine {
2
3     public static void one() {
4         System.out.print("one");
5     }
6
7     public static void main(String[] args) {
8         System.out.println(one() + 7);
9     }
10 }
```

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



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

Powered by  trinket  
one7

# Write a `isDivisible()` function

- Hint: use modulus

NewLine.java

```
1 public class NewLine {
2
3     public static boolean isDivisible(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(isDivisible(10,2));
12        System.out.println(isDivisible(10,3));
13        System.out.println(isDivisible(12,3));
14        System.out.println(isDivisible(12,5));
15    }
16 }
```

Powered by  trinket

true	
false	
true	
false	

# Write a `isDivisible()` function

- A better version

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

- Title Bar:** The title bar displays "NewLine.java" next to a gear icon.
- Code Area:** The main area contains the following Java code:

```
1 public class NewLine {  
2     public static boolean isDivisible(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    public static void main(String[] args) {  
11        System.out.println(isDivisible(10,2));  
12        System.out.println(isDivisible(10,0));  
13        System.out.println(isDivisible(10,3));  
14    }  
15 }  
16 }
```
- Right Panel:** On the right side, there is a vertical toolbar with a plus sign and an upward arrow icon.
- Vertical Ruler:** A vertical ruler is visible on the far right edge of the editor.

# 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
public class NewLine {
    public static boolean isTriangle(int a, int b, int c) {
        if(c > a + b)
            return false;
        else if(b > a + c)
            return false;
        else if(a > b + c)
            return false;
        else
            return true;
    }
    public static void main(String[] args) {
        System.out.println(isTriangle(3,4,5));
        System.out.println(isTriangle(3,4,8));
        System.out.println(isTriangle(12,3,11));
        System.out.println(isTriangle(12,3,5));
    }
}
```

Powered by  trinket

true
false
true
false