LEARN PROGRAMMING IN JAVA™

Learning OO Java™ as a Beginner – Operators

COURSE AGENDA

- What is an Object
- The Basics
- Operators
- Control Flow
- Implementation Hiding
- Reuse
- Interfaces
- Collections
- Functional Programming
- Stream
- Exceptions
- Generics
- Arrays

RECOMMENDED TEXTS

- Java 5 Book
  - Thinking in Java – 4th Edition - Free – PDF - GitHub
  - Thinking in Java – 4th Edition – Hard Cover - Amazon
- Recommended Java 8 Book
  - Bruce Eckel on Java 8
    - Contains references to newer Java syntax
- #1 Best Seller in Beginner's Guide to Java Programming
  - Head First Java: A Brain-Friendly Guide 3rd Edition

REQUIRED SOFTWARE

- JDK™
  - JDK 8 or Newer (Open Source, Amazon, or Oracle)
  - The beginner course focuses on core language constructs; feel free to use 11, 17, 21, etc.

- IDE
  - IntelliJ Community Edition (Free) or Enterprise (Paid)
  - While you can use any IDE with Java, the course and labs are explicitly made with IntelliJ in mind

INSTRUCTOR – HUGO SCAVINO

- 30 Years of IT Experience
- Using Java since the beta
- Taught Java and OOP in USA, UK, and France to Fortune 500
- Senior Software Architect with
  - DTCC
  - Penske
  - HSBC
  - Government Agencies

https://www.linkedin.com/in/hugoscavino/

TOPIC DESCRIPTION

Operators are special symbols and keywords categorized into different types based on their functionality.

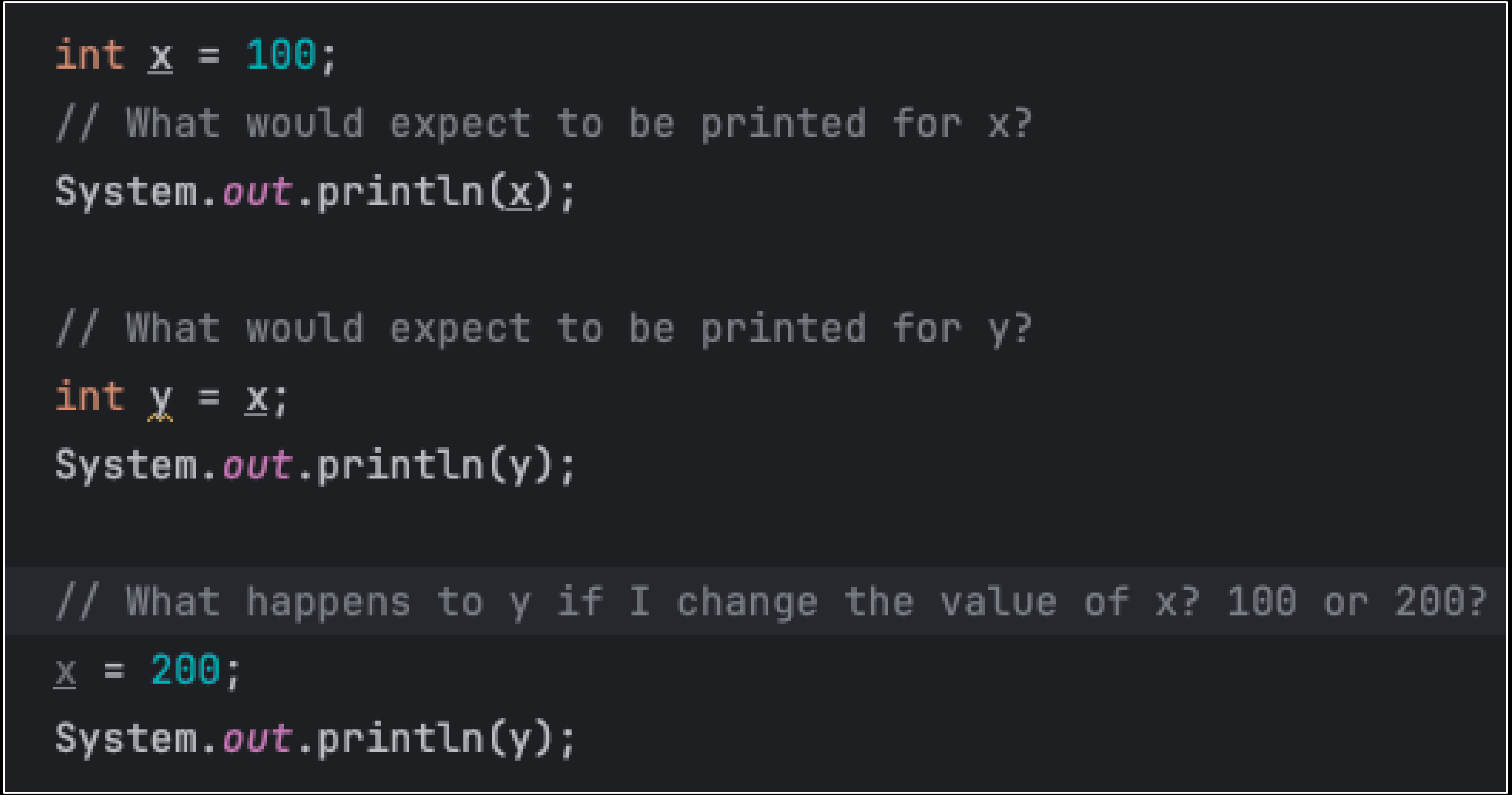
- Arithmetic operators (+, -, *, /, %) performing basic math operations.
- Relational operators (==, !=, <, >, <=, >=) compare two values.
- Logical operators (&&, | |, !) combine or negate boolean expressions.
- Assignment operators (=, +=, -=, etc.) assign values to variables
  Unary operators (e.g., +, -, ++, --) work on a single operand to perform tasks like incrementing or negating.

ASSIGNMENT

- Assigning a field to a value copies the value to it
- If you assign a variable a literal, it is assigned with that value until it is changed.
  - `int x = 100;`
  - The value of `x` equals 100

- If you assign a variable a value from another value, it is assigned with that value until it is changed.
  - `int y = x;`
  - The value of `y` is the same value of `x,` which is 100 in this case

ASSIGNMENT EXAMPLE

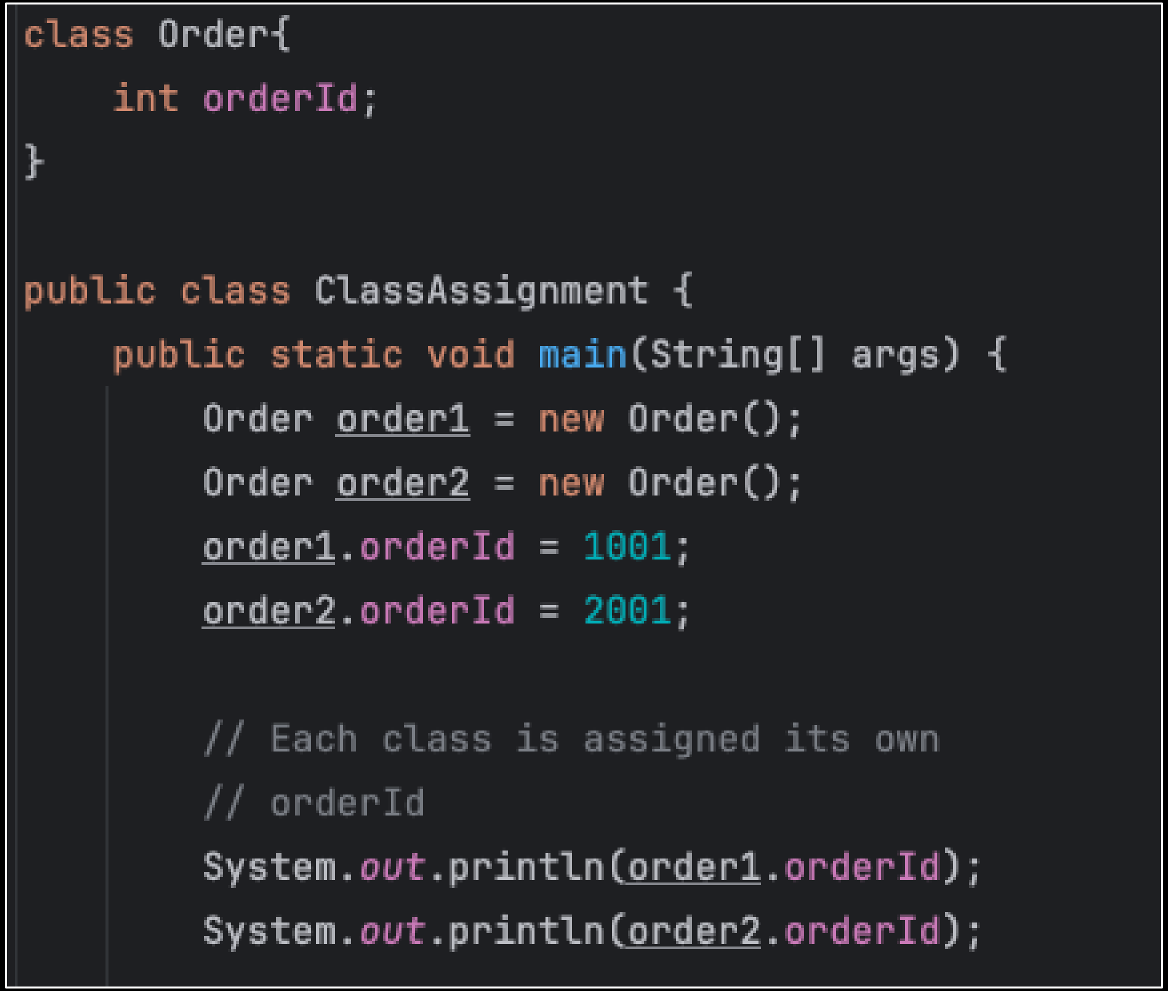


```
int x = 100;
// What would expect to be printed for x?
System.out.println(x);

// What would expect to be printed for y?
int y = x;
System.out.println(y);

// What happens to y if I change the value of x? 100 or 200?
x = 200;
System.out.println(y);
```

ASSIGNMENT W/ CLASSES

- What happens to the fields of two objects if you assign the classes to each other?
- What if I say ClassA is equal to ClassB?
- What happens to the fields in each class?
- If I change the field value in one class? (e.g., ClassA)
- Does that change the value in the other class? (e.g., ClassB)

ASSIGNMENT W/ CLASSES EXAMPLE

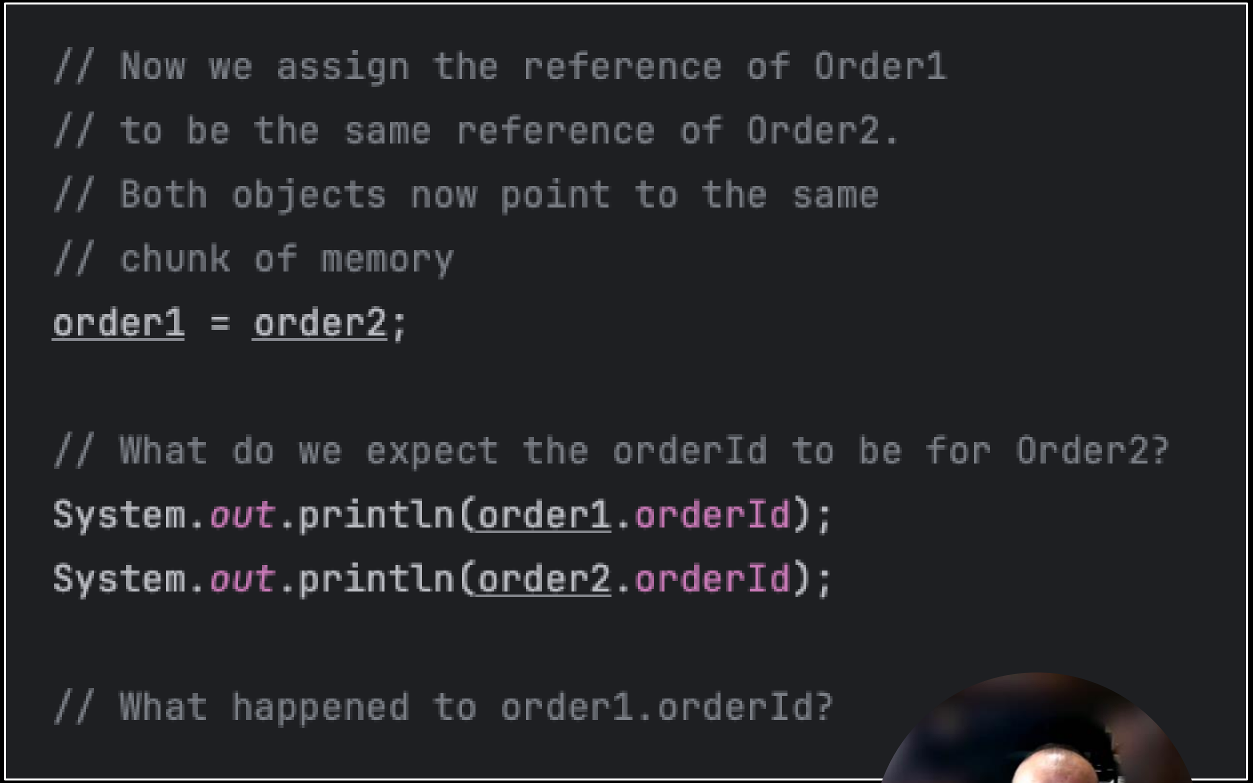


```java
class Order{
    int orderId;
}

public class ClassAssignment {
    public static void main(String[] args) {
        Order order1 = new Order();
        Order order2 = new Order();
        order1.orderId = 1001;
        order2.orderId = 2001;

        // Each class is assigned its own
        // orderId
        System.out.println(order1.orderId);
        System.out.println(order2.orderId);
```

These assignments are what you may expect.

Create one object, assign it a value, then create another and give it its value.

EXAMPLE – PT2



```
// Now we assign the reference of Order1
// to be the same reference of Order2.
// Both objects now point to the same
// chunk of memory
order1 = order2;

// What do we expect the orderId to be for Order2?
System.out.println(order1.orderId);
System.out.println(order2.orderId);

// What happened to order1.orderId?
```

What happens when you assign the first object the same value as the other object?

(NOT THE FIELDS THE OBJECTS)

Is this what you expected? What happened to the value of `order1`?

`order1.orderId` now get the value from `order2.orderId` which is `2001`

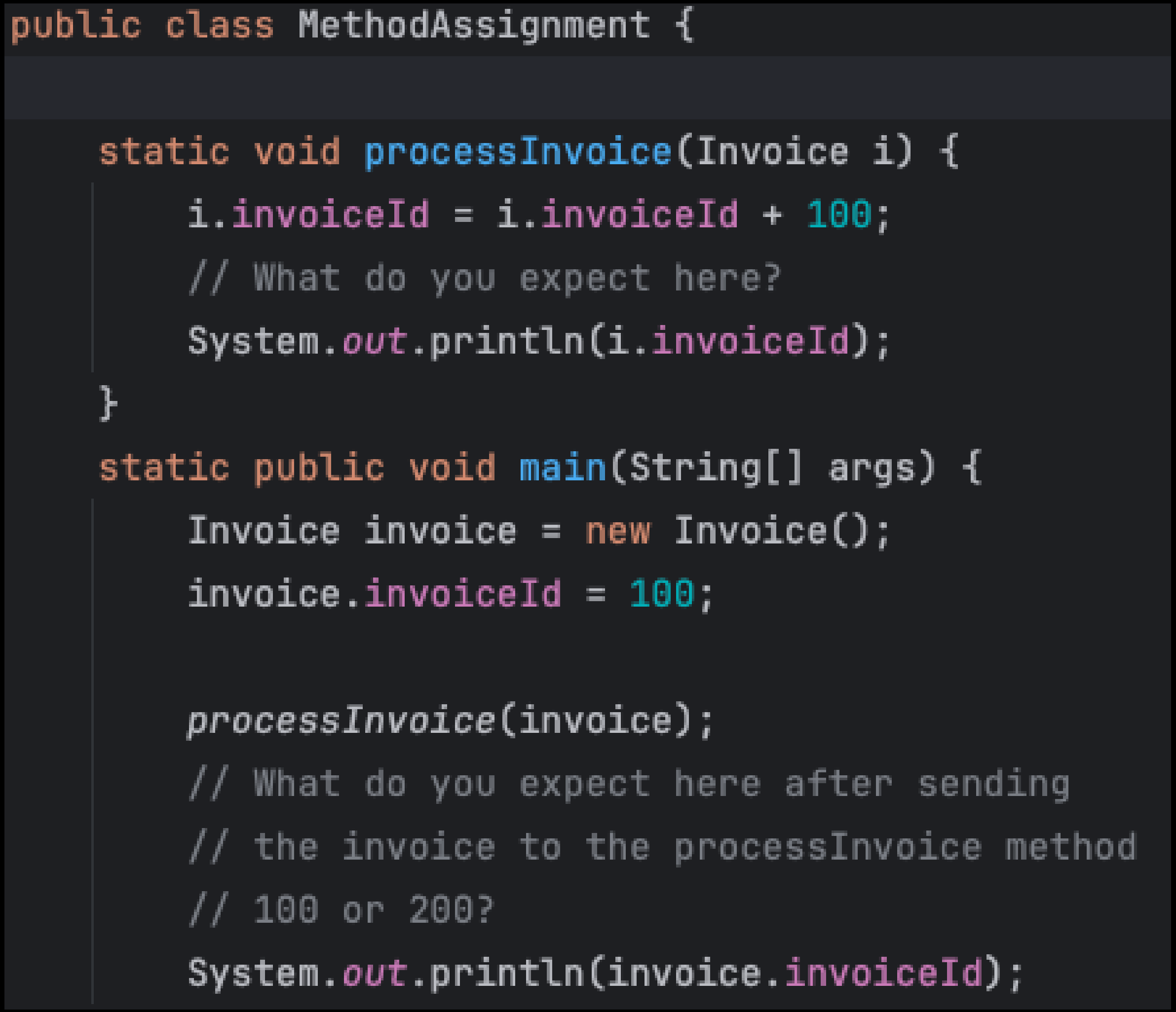
ASSIGNMENT IN METHODS

- What if you pass an object to a method and change the field's value in that method?

- What is the value of the object's field in the method?
- What is the value of the object's field after the method changes it?

- e.g., Create an instance of the Invoice class. Set the `invoiceId` to 100
  - Send that object to a method that operates on an `Invoice` class
  - The method changes the `invoiced;` what is the value?
  - What is the value of `invoiceId` when the object is referenced after being passed into the method?

ASSIGNMENT W/ METHODS EXAMPLE



```
public class MethodAssignment {

    static void processInvoice(Invoice i) {
        i.invoiceId = i.invoiceId + 100;
        // What do you expect here?
        System.out.println(i.invoiceId);
    }
    static public void main(String[] args) {
        Invoice invoice = new Invoice();
        invoice.invoiceId = 100;

        processInvoice(invoice);
        // What do you expect here after sending
        // the invoice to the processInvoice method
        // 100 or 200?
        System.out.println(invoice.invoiceId);
```

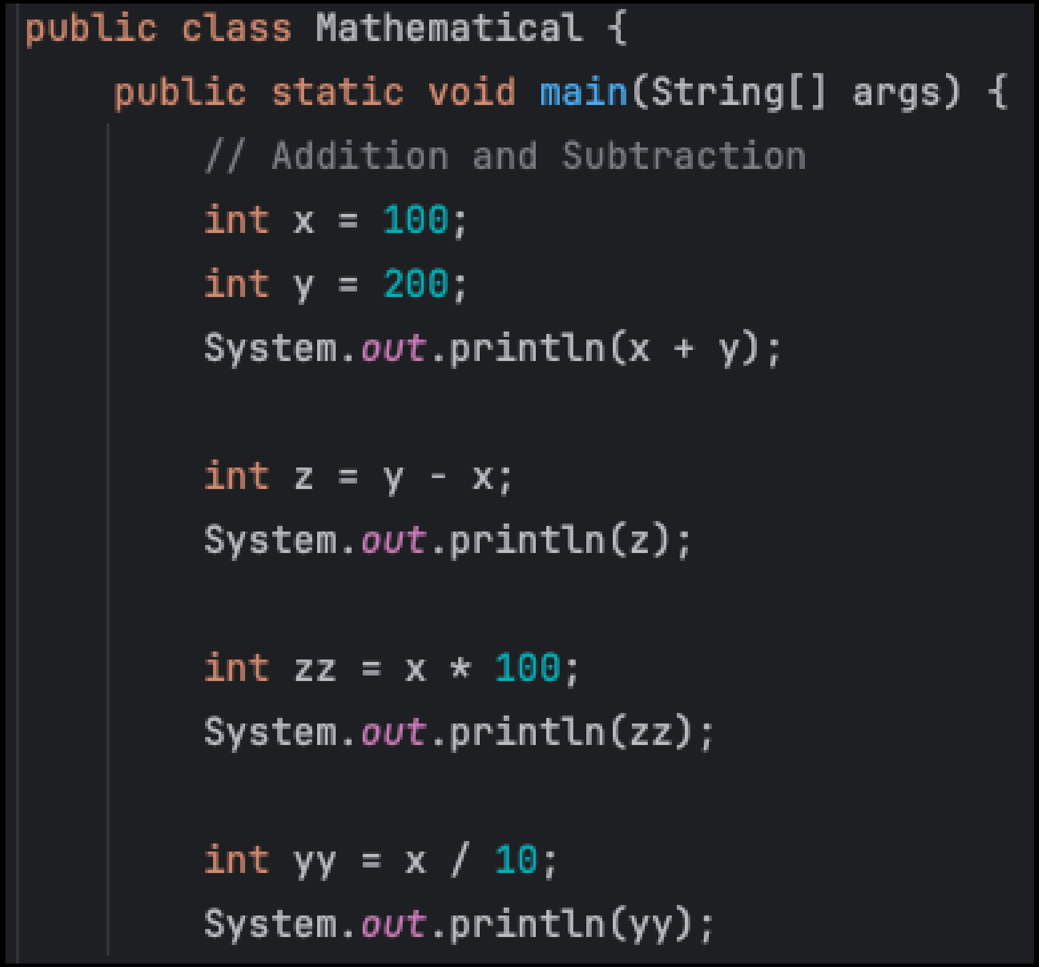
You expect the `invoiceId` with the `processInvoice` to add 100 to the `invoiceId`.

Why did the `invoiceId` become the changed value?

Why did the value not remain the initial 100?

You are NOT making a copy; you are modifying the same object!

MATHEMATICAL OPERATORS

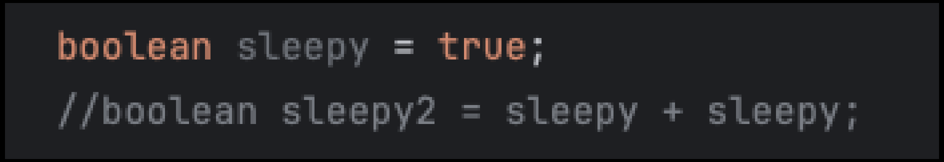


```java
public class Mathematical {
    public static void main(String[] args) {
        // Addition and Subtraction
        int x = 100;
        int y = 200;
        System.out.println(x + y);

        int z = y - x;
        System.out.println(z);

        int zz = x * 100;
        System.out.println(zz);

        int yy = x / 10;
        System.out.println(yy);
```

As you would expect, there is addition and subtraction.

There is multiplication and division for each primitive type, `int`, `float`, `long`, `short`, `byte`, etc

But not for `boolean`



```java
boolean sleepy = true;
//boolean sleepy2 = sleepy + sleepy;
```

AUTO INCREMENT AND DECREMENT

The auto-increment and auto-decrement features allow variables, usually integers, to be incremented or decremented by 1 automatically. These operations use the ++ (increment) and -- (decrement) operators.

```
int x = 100;
// z equals 100,
// x then becomes 101
int z = x++;
```

```
int k = 100;
// j equals 100,
// then k = 99
int j = k--;
```

AUTO PRE-INCREMENT AND DECREMENT

The pre-increment and decrement perform the operation first then the assignment

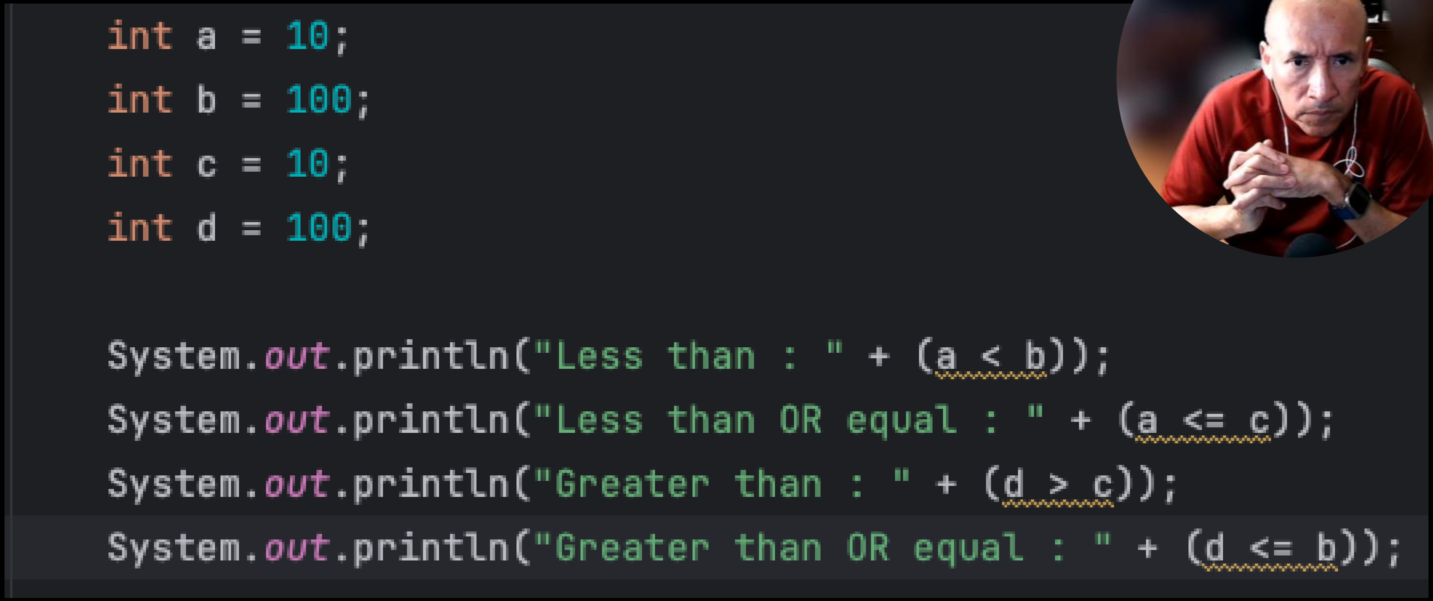
```
int x = 100;
// x equals 101,
// z is then 101
int z = ++x
```

```
int k = 100;
// y equals 99,
// then k = 99
int j = --k;
```

RELATIONAL OPERATORS

- There are six.
  - They ALL produce a `true` or `false` result. There is no middle ground.

- Less than '<'
- Greater than '>'
- Less Than OR Equal '<='
- Greater Than OR Equal '>='
- Equals is `==`
- Not Equals is `!=`



```java
int a = 10;
int b = 100;
int c = 10;
int d = 100;

System.out.println("Less than : " + (a < b));
System.out.println("Less than OR equal : " + (a <= c));
System.out.println("Greater than : " + (d > c));
System.out.println("Greater than OR equal : " + (d <= b));
```

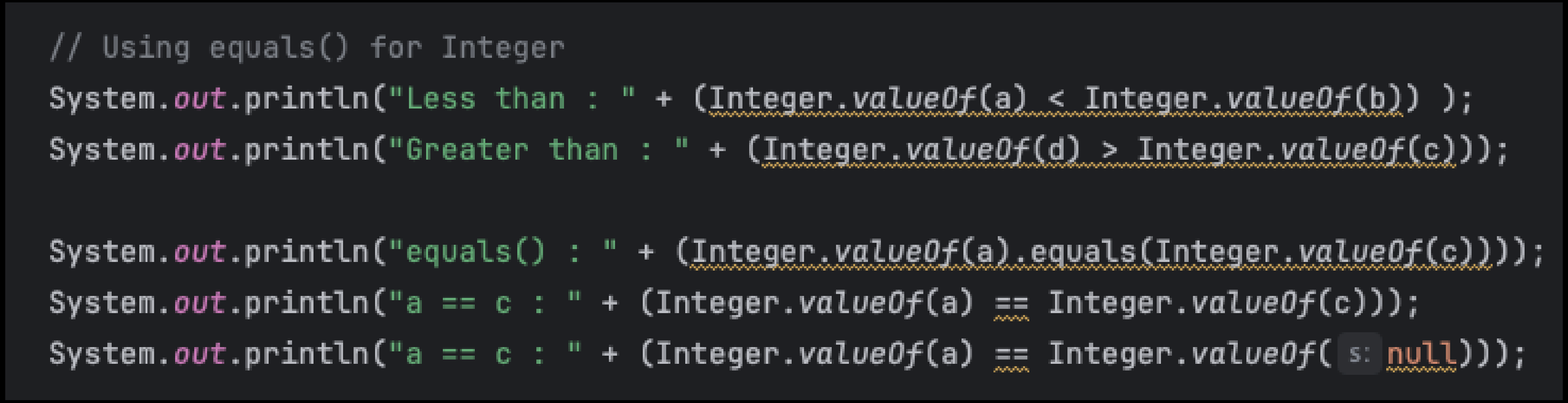
MORE ON OPERATORS

- For `Integer` you will use the relational operators and the equals() method.

- Less than ‘<‘
- Greater than ‘>’
- equals()

- Don’t use `==` with Integer, or you will get the below warning and issues with `null`

- `Warning! Number objects are compared using '==', not 'equals()'`



```java
// Using equals() for Integer
System.out.println("Less than : " + (Integer.valueOf(a) < Integer.valueOf(b)) );
System.out.println("Greater than : " + (Integer.valueOf(d) > Integer.valueOf(c)));

System.out.println("equals() : " + (Integer.valueOf(a).equals(Integer.valueOf(c))));
System.out.println("a == c : " + (Integer.valueOf(a) == Integer.valueOf(c)));
System.out.println("a == c : " + (Integer.valueOf(a) == Integer.valueOf( s: null)));
```

COMPARING OBJECTS USING EQUALS()

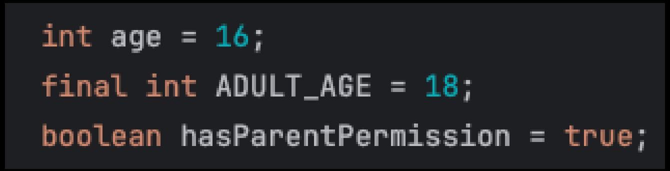
equals()

- There are complicated steps to use equals() for your custom classes.
- You need to make sure you are comparing contents and not references to objects
- Later in the course….

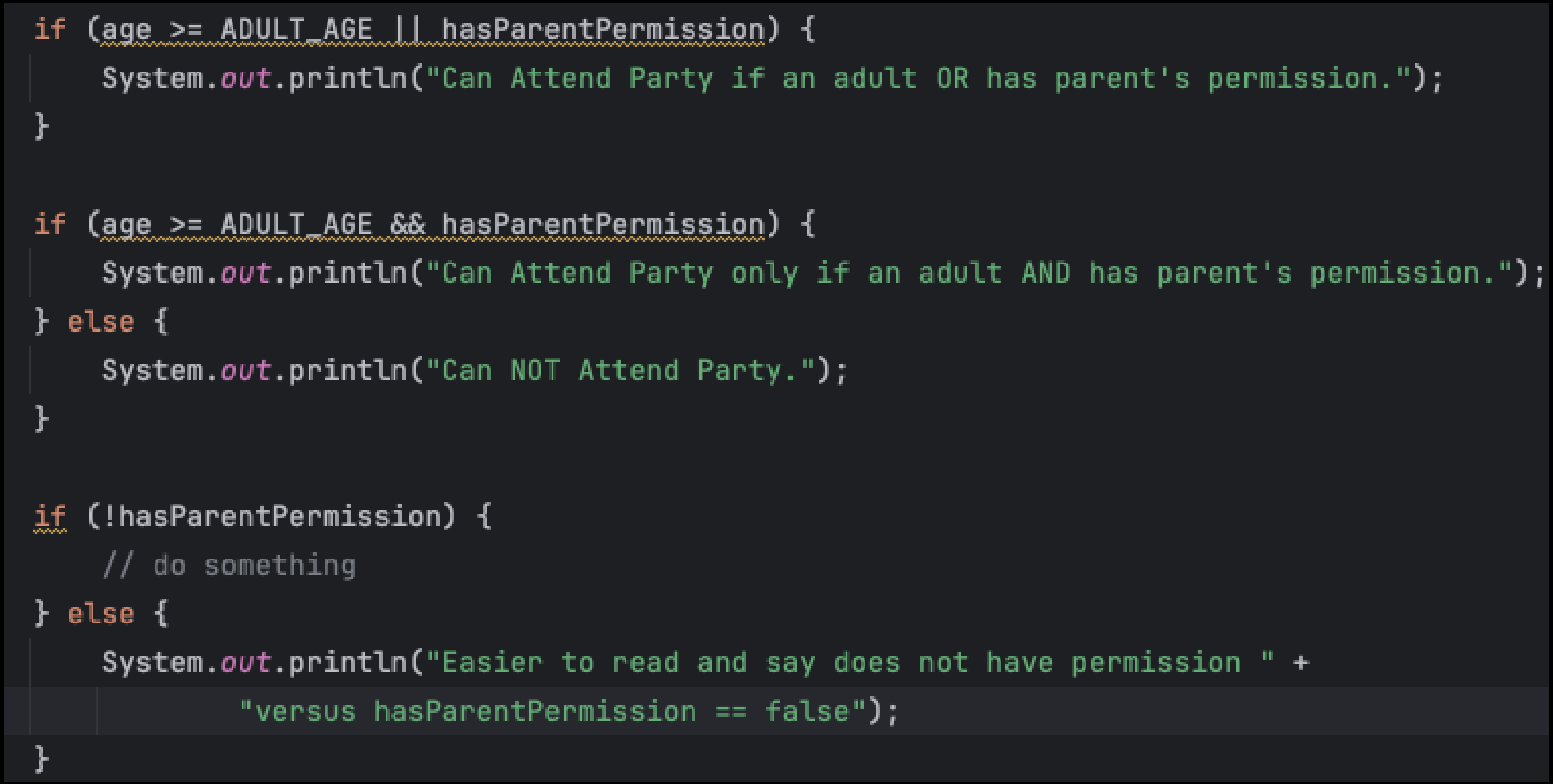
LOGICAL OPERATORS

- Logical operators perform operations on `boolean` expressions, returning either `true` or `false`. They are typically used in decision-making statements such as if, while, and for loops.
- AND (&&)
  - All the expressions MUST be true for the entire expression to be true.
- OR (| |)
  - For the entire expression to be true, at LEAST ONE expression MUST be true.
- NOT (!)
  - Negates the `boolean` value. Great for readability

LOGICAL OPERATORS - EXAMPLE



```java
int age = 16;
final int ADULT_AGE = 18;
boolean hasParentPermission = true;
```



```java
if (age >= ADULT_AGE || hasParentPermission) {
    System.out.println("Can Attend Party if an adult OR has parent's permission.");
}

if (age >= ADULT_AGE && hasParentPermission) {
    System.out.println("Can Attend Party only if an adult AND has parent's permission.");
} else {
    System.out.println("Can NOT Attend Party.");
}

if (!hasParentPermission) {
    // do something
} else {
    System.out.println("Easier to read and say does not have permission " +
            "versus hasParentPermission == false");
}
```

What is

short circuiting?

Will

`hasParent`
`Permission`

be evaluated?

BITWISE AND SHIFT OPERATORS

These are dumb and will not be covered in this course.

TERNARY

- Ternary operator is a shorthand for `if-else` statements. It allows you to write concise conditional expressions in a **single line**. The ternary operator is often used when assigning a value or executing a simple logic based on a condition.
  - It is not readable; dubious use

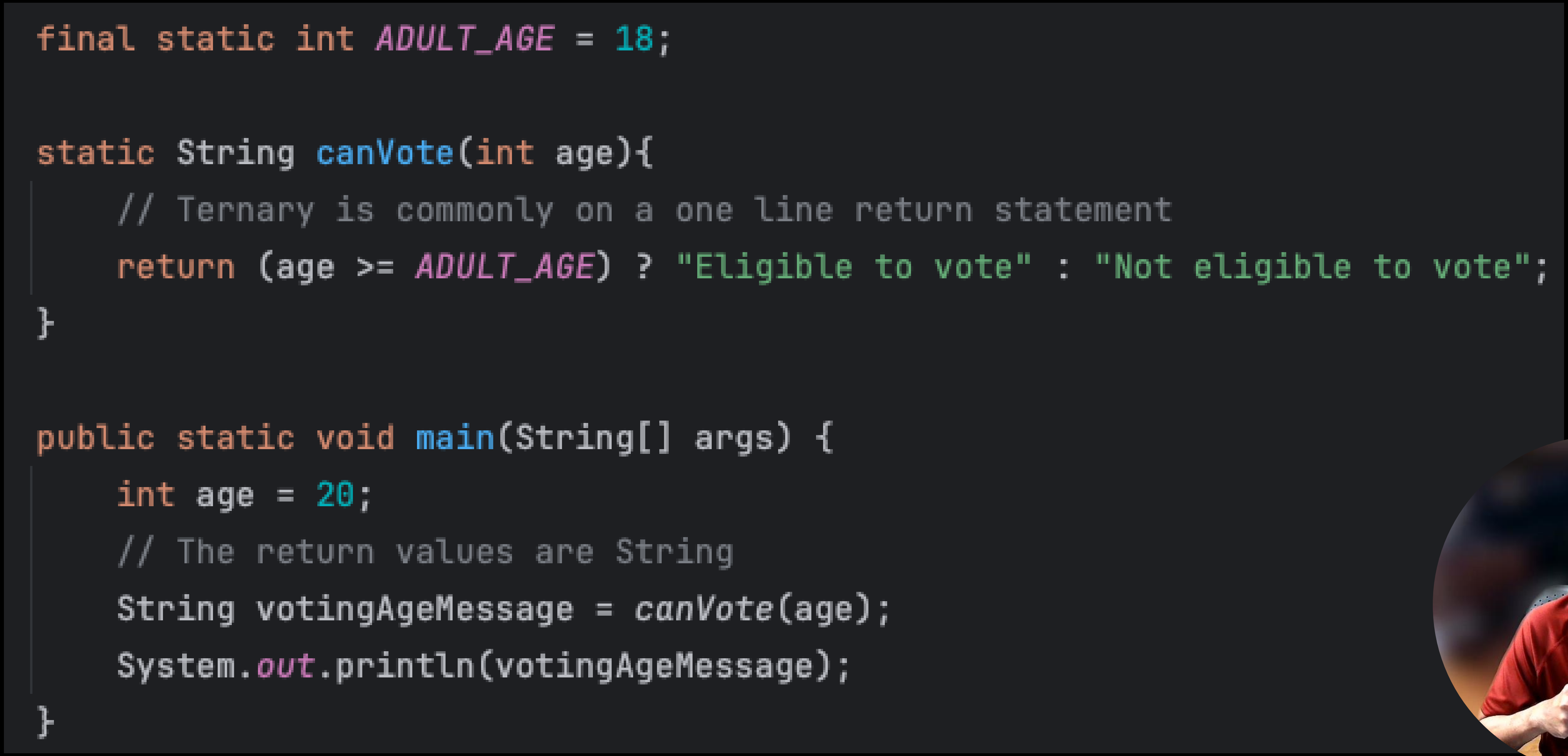
```
variable = (condition) ? expression1 : expression2;
```

- condition: boolean expression that evaluates to either true or false.

- expression1: Executed if the condition is true.
- expression2: Executed if the condition is false.

TERNARY - EXAMPLE

```
variable = (condition) ? expression1 : expression2;
```



```java
final static int ADULT_AGE = 18;

static String canVote(int age){
    // Ternary is commonly on a one line return statement
    return (age >= ADULT_AGE) ? "Eligible to vote" : "Not eligible to vote";
}

public static void main(String[] args) {
    int age = 20;
    // The return values are String
    String votingAgeMessage = canVote(age);
    System.out.println(votingAgeMessage);
}
```

TOPIC SUMMARY

- Arithmetic operators (+, -, *, /, %) performing basic math operations.
- Relational operators (==, !=, <, >, <=, >=) compare two values.
- Logical operators (&&, | |, !) combine or negate boolean expressions.
- Assignment operators (=, +=, -=, etc.) assign values to variables
  Unary operators (e.g., +, -, ++, --) work on a single operand to perform tasks like incrementing or negating.