# Workshop on: Java Fundamentals (J2SE)

Rahul Dhangar • Day 1

# Overview

## Duration

- Four Days

## Prerequisites

- Install IntelliJ IDEA (Community Edition)

## Takeaways

- stay tuned...

## Day 1 - Topics

- Writing Our First Java Program
- Decision Structures
- Loops
- Methods

## Day 2 - Topics

- Classes and Objects
- Arrays
- Text processing

## Day 3 - Topics

- Inheritance
- Polymorphism
- Abstraction
- Interfaces

## Day 4 - Topics

- Data Structures
- Iterating Data Structures
- Functional Interfaces
- Streams
- Exception Handling

## Key Takeaways!

1. Learn basics of Java programming language
2. Practice both object-oriented and functional programing
3. Gain exposure to modern Java features
4. Exercise concepts with fun coding challenges

## Let’s start!

## Setup steps:

1) **Install IntelliJ Community Edition**

   https://www.jetbrains.com/idea/download/

2) **Java 17 or later**

   IntelliJ > Download JDK

3) **GitHub repo**

   https://github.com/rahuldhangar/mandsaur-university/

# IntelliJ IDEA

## New Project > Maven

- Powerful project management tool that is based on POM (project object model). Used for projects build, dependency and documentation.

- Helps in developing reports, checks build and testing automation setups.

# The Basics

let's write our first program to learn the basics

## P0: Gross Pay Calculator

*Write a program that calculates an employee's gross pay*

*We will learn:*

- packages
- classes
- main method
- input
- output

## Questions ?

What happens if we don’t close scanner ?

## Decision Structures

programs often have to evaluate conditions to determine which blocks of code to execute

## Decision Structures

- Types of Decision Structures
- Relational & Logical Operators

## Decision Structures

- Types of Decision Structures

1) If Statements

If a certain situation occurs, do <something>,

Then go back to the main flow

## P1: Salary Calculator

*All salespeople get a payment of $ 1000 for the week.*

*Salespeople who exceed 10 sales get an additional bonus of $ 300*

## Decision Structures

- Types of Decision Structures

2) If-Else Statements

   If a certain situation occurs, <do something>.

   Otherwise <do something else>.

## P2: Quota Calculator

*All salespeople are expected to make at least 10 sales each week.*

*For those who do, they receive a congratulatory message.*

*For those who don’t, they are informed of how many sales they were short.*

## Decision Structures

- Types of Decision Structures

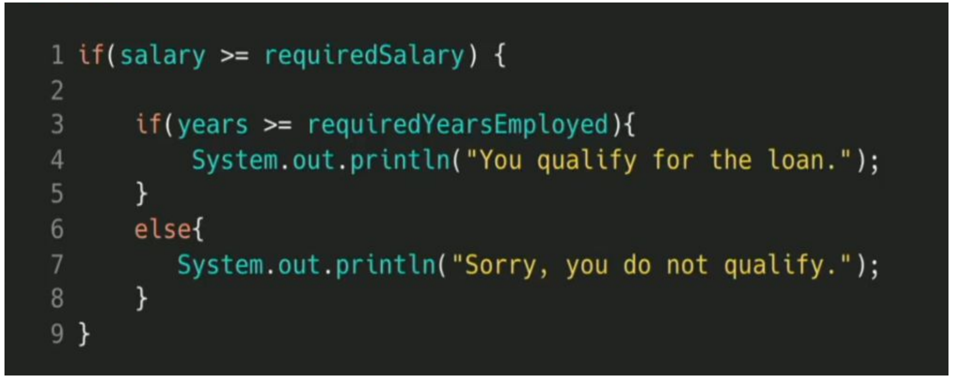
3) a] Nested If Statements

Path inside of a path
If a certain situation occurs, check for the next situation

b] If-Else-If Statement

If situation A occurs, <do something>
Else if situation B occurs, <do something else>
Else if situation C occurs, <do something else>

## Example: Nested If Statements



```
if(salary >= requiredSalary) {

    if(years >= requiredYearsEmployed){
        System.out.println("You qualify for the loan.");
    }
    else{
       System.out.println("Sorry, you do not qualify.");
    }
}
```

## P3: Test Results

*Display a letter grade for a student based on their test score*

## Decision Structures

- Types of Decision Structures

4) Switch Statements

Solves problem in the same way that if-else-if does

Good for when there is more than 2 possible paths

Each path checks for equality

## P4: Grade Message

*Have a user enter their letter grade, and using a switch statement, print out a message letting them know how they did.*

## Decision Structures

- Types of Decision Structures

5) Switch Expressions

   Similar to switch statements but allow you to directly assign a value when a case is matched.

## P5: Grade Message

*Have a user enter their letter grade, and using a switch expression, print out a message letting them know how they did.*

## Questions ?

Let’s take a break :)

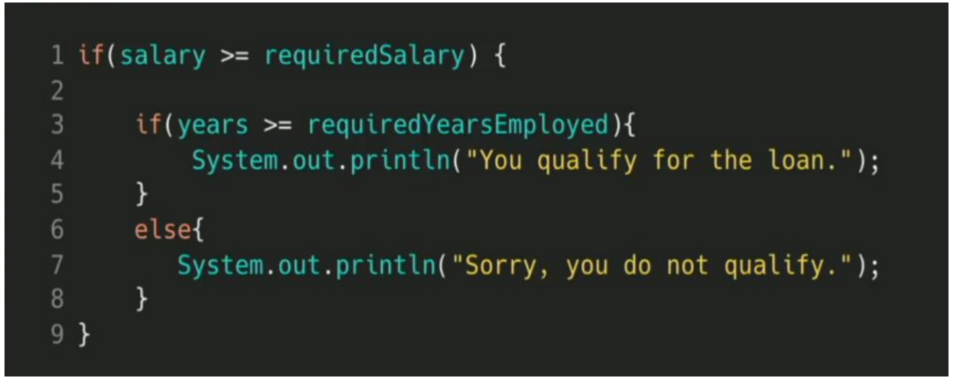
## Relational Operators

| OPERATOR | MEANING | EXAMPLE |
|---|---|---|
| > | Greater than | 2 > 3 is false |
| < | Less than | 2 < 3 is true |
| >= | Greater than or equal to | 4 >= 4 is true |
| <= | Less than or equal to | 4 <= 3 is false |
| == | Equal to | 3 == 2 is false |
| != | Not equal to | 3 != 2 is true |

## Logical Operators

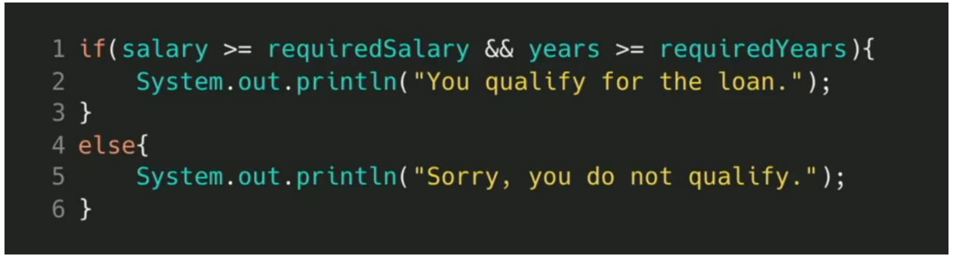
| SYMBOL | OPERATOR | MEANING | EXAMPLE |
|---|---|---|---|
| && | AND | Both conditions must be true | 1 <= 2 && 4 != 5 |
| \|\| | OR | At least one condition must be true | 3 == 4 \|\| 2 == 2 |
| ! | NOT | Condition must be false | !( 2 == 3) |

## Previously: Nested If Statements:



```java
if(salary >= requiredSalary) {

    if(years >= requiredYearsEmployed){
        System.out.println("You qualify for the loan.");
    }
    else{
       System.out.println("Sorry, you do not qualify.");
    }
}
```

## Using Logical Operators:



```java
if(salary >= requiredSalary && years >= requiredYears){
    System.out.println("You qualify for the loan.");
}
else{
    System.out.println("Sorry, you do not qualify.");
}
```

## Repetition Structures

Loops are structures that causes a block of code to repeat.

## P6: Input Validation

*Each store employee makes $15 an hour. Write a program that allows the employee to enter the number of hours worked for the week. Do not allow overtime.*

## While Loops

**Condition Controlled**

Continues running while the specified condition remains true

**Pre-test**

Condition is tested before entering the loop

**Execution**

Use when loop may or may not need to be executed

## P7: Add Numbers

*Write a program that allows a user to enter two numbers, and then sums up the two numbers. The user should be able to repeat this action until they indicate they are done.*

## Do While Loops

**Condition Controlled**

Continues running while the specified condition remains true

**Pre-test**

Condition is tested **after** entering the loop

**Execution**

Use when loop should run at least once, and possibly more

## P8: Cashier

*Write a cashier program that will scan a given number of items and tally the cost.*

# For Loops

**Count Controlled**

Runs a specified number of times

**Pre-test**

Condition is tested before entering the loop

**Execution**

Use when you know how many times the loop should be executed

## Nested Loops

Sometimes your repetitive tasks also contain repetitive subtasks

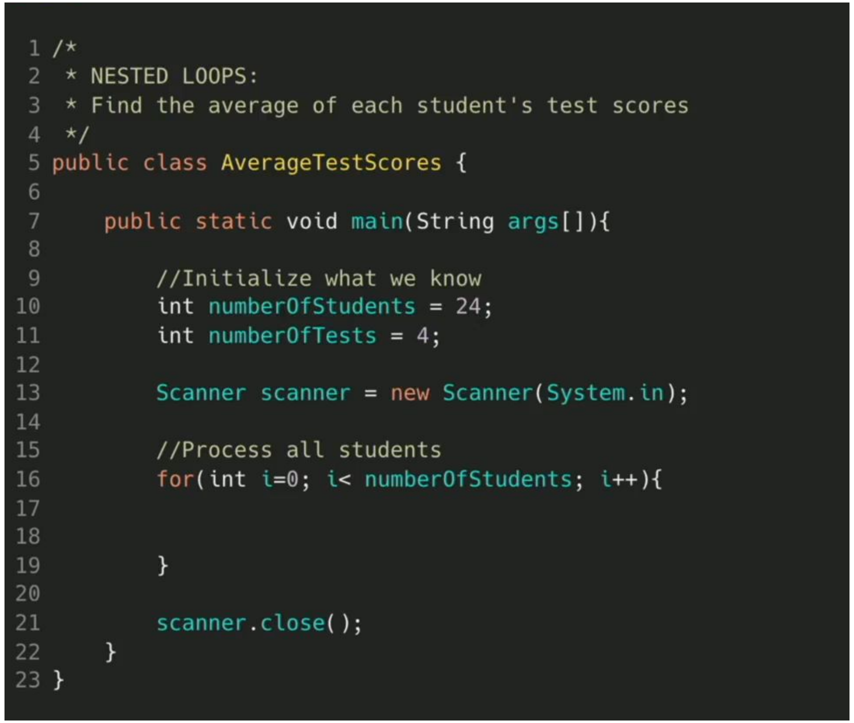
## P9: Nested Loops Example

*Find the average test scores for each student in the class.*

**Loop 1:** go through every student in the class

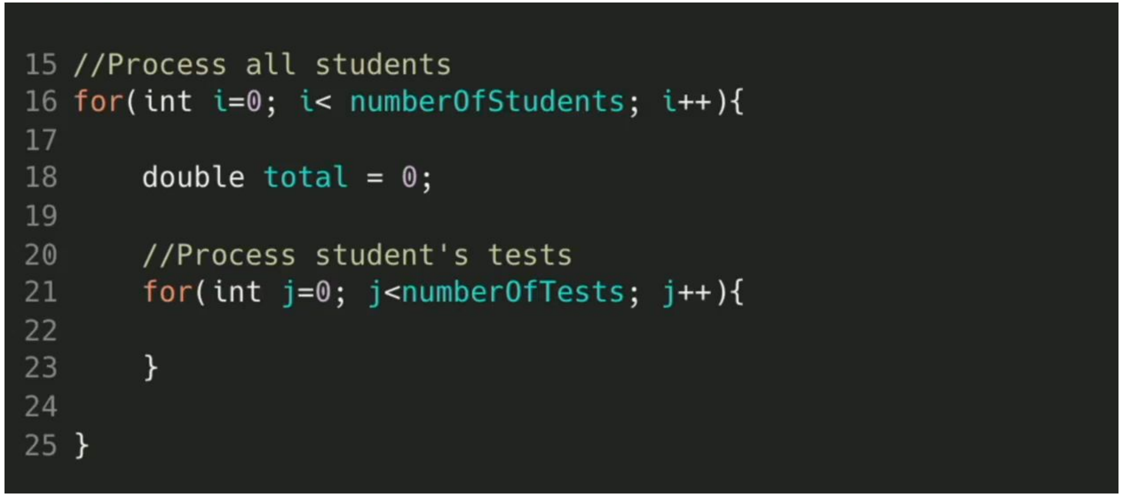
**Loop 2:** go through every test grade that student has

# P9: Nested Loops Example



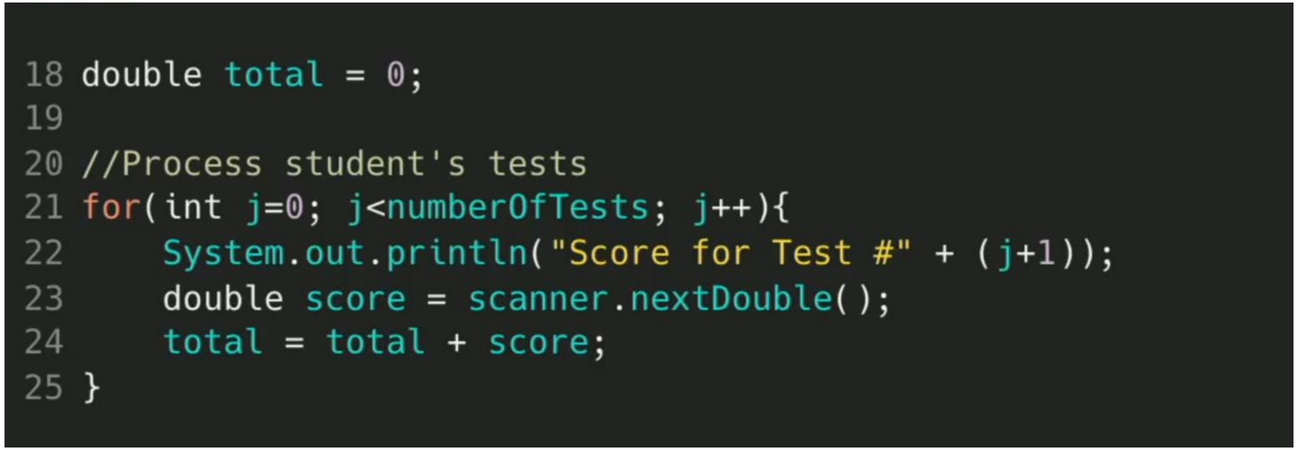
```java
/*
 * NESTED LOOPS:
 * Find the average of each student's test scores
 */
public class AverageTestScores {

    public static void main(String args[]){

        //Initialize what we know
        int numberOfStudents = 24;
        int numberOfTests = 4;

        Scanner scanner = new Scanner(System.in);

        //Process all students
        for(int i=0; i< numberOfStudents; i++){


        }

        scanner.close();
    }
}
```

## P9: Nested Loops Example



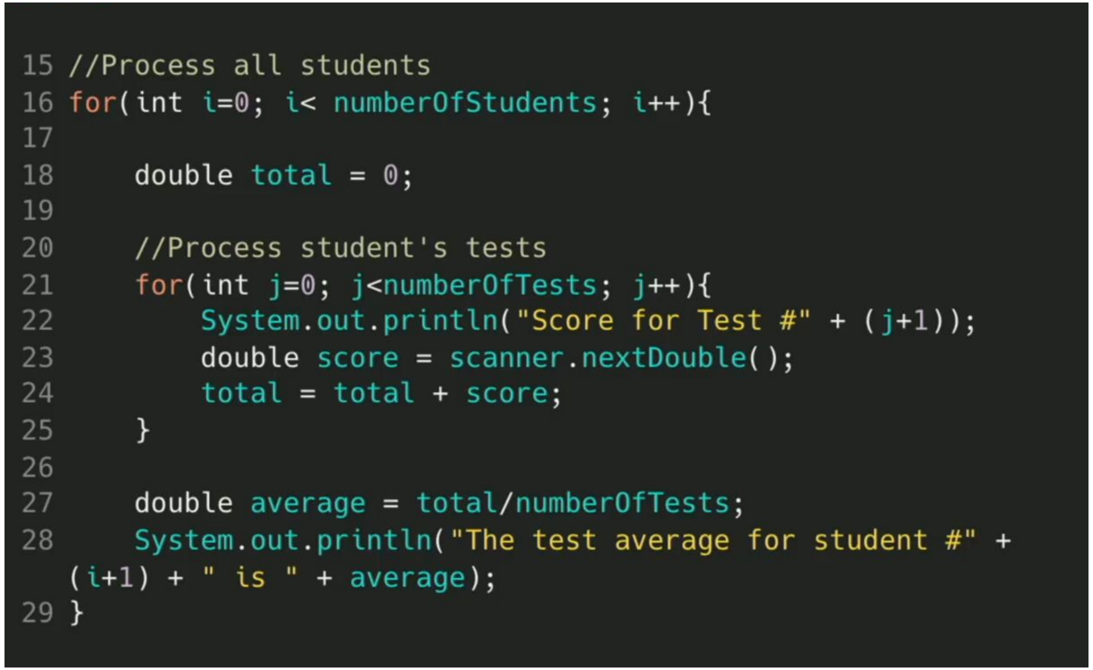
```
//Process all students
for(int i=0; i< numberOfStudents; i++){

    double total = 0;

    //Process student's tests
    for(int j=0; j<numberOfTests; j++){

    }

}
```

## P9: Nested Loops Example



```
double total = 0;

//Process student's tests
for(int j=0; j<numberOfTests; j++){
    System.out.println("Score for Test #" + (j+1));
    double score = scanner.nextDouble();
    total = total + score;
}
```

# P9: Nested Loops Example



```java
//Process all students
for(int i=0; i< numberOfStudents; i++){

    double total = 0;

    //Process student's tests
    for(int j=0; j<numberOfTests; j++){
        System.out.println("Score for Test #" + (j+1));
        double score = scanner.nextDouble();
        total = total + score;
    }

    double average = total/numberOfTests;
    System.out.println("The test average for student #" + (i+1) + " is " + average);
}
```

## Methods

Collection of statements that perform a task

Things that breaks complex programs into small manageable pieces

## Methods

```
public static int calculateSum( int number1, int number2) {

    int sum = number1 + number2;
    return sum;
}
```

## P10: Greet User

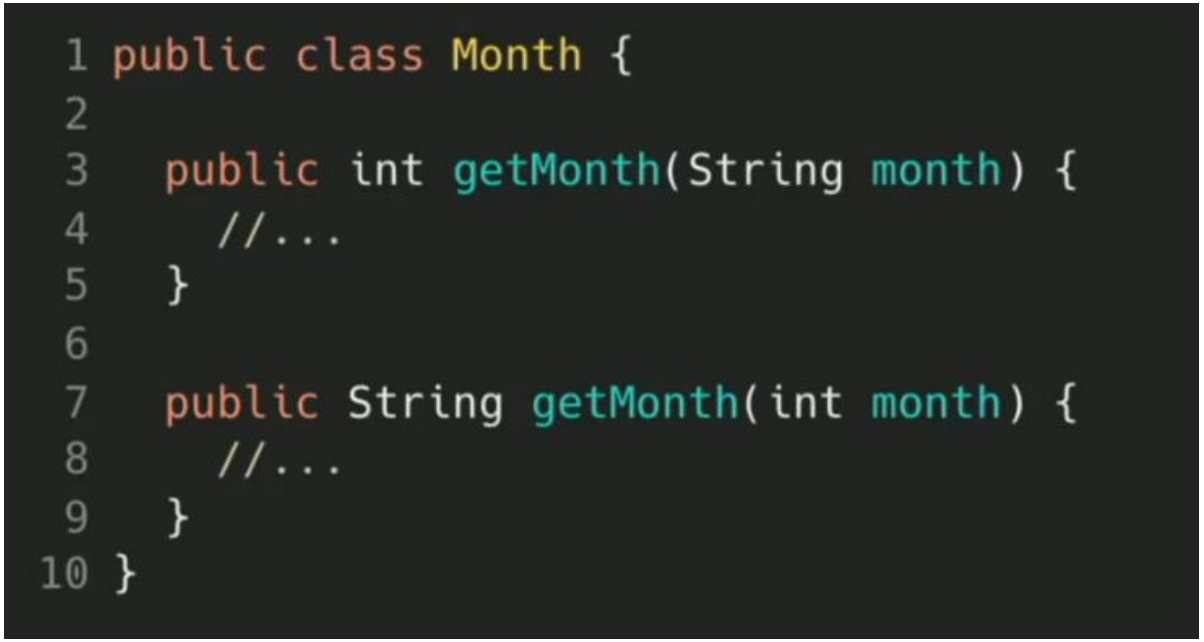
*Write a method that asks a user for their name, then another method that greets the user by name.*

## Overloaded Methods

A class can have multiple methods with the same name, but

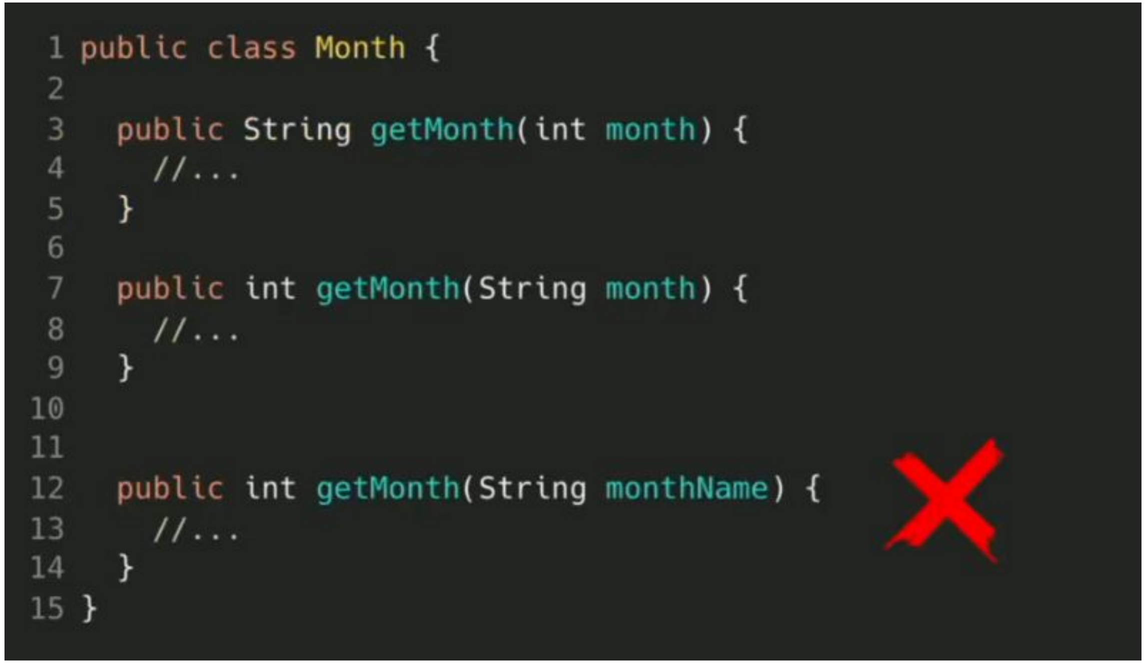
they must have unique parameter lists

## Method Overloading



```java
public class Month {

  public int getMonth(String month) {
    //...
  }

  public String getMonth(int month) {
    //...
  }
}
```

## Method Overloading



```
public class Month {

  public String getMonth(int month) {
    //...
  }

  public int getMonth(String month) {
    //...
  }


  public int getMonth(String monthName) {
    //...
  }
}
```

## Variable Scope

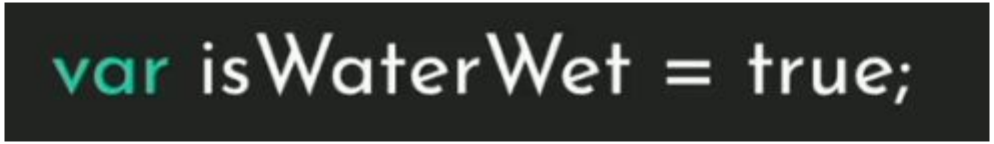
*A variable is only accessible within the scope it is declared within*

## P11: Instant User Check

*Write a program that approves anyone who makes more than $25,000 and has a credit score of 700 or better. Reject all others.*

## Local Variable Type Inference

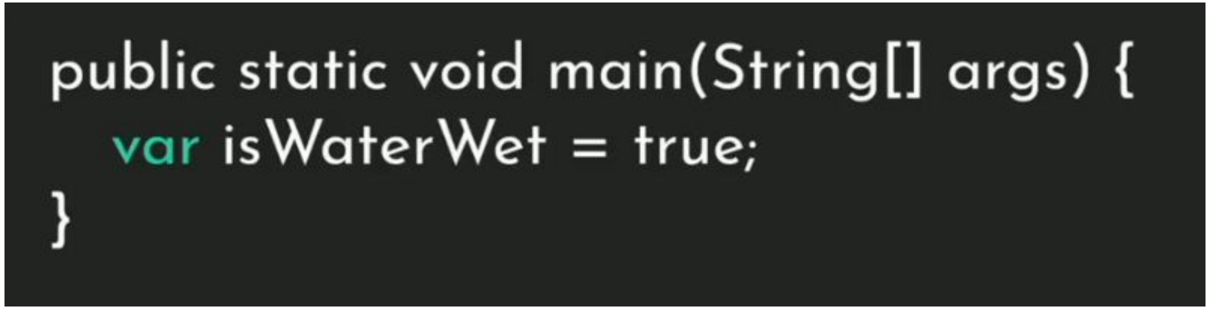
*When declaring a local variable as var, Java will infer its type.*



```
var isWaterWet = true;
```

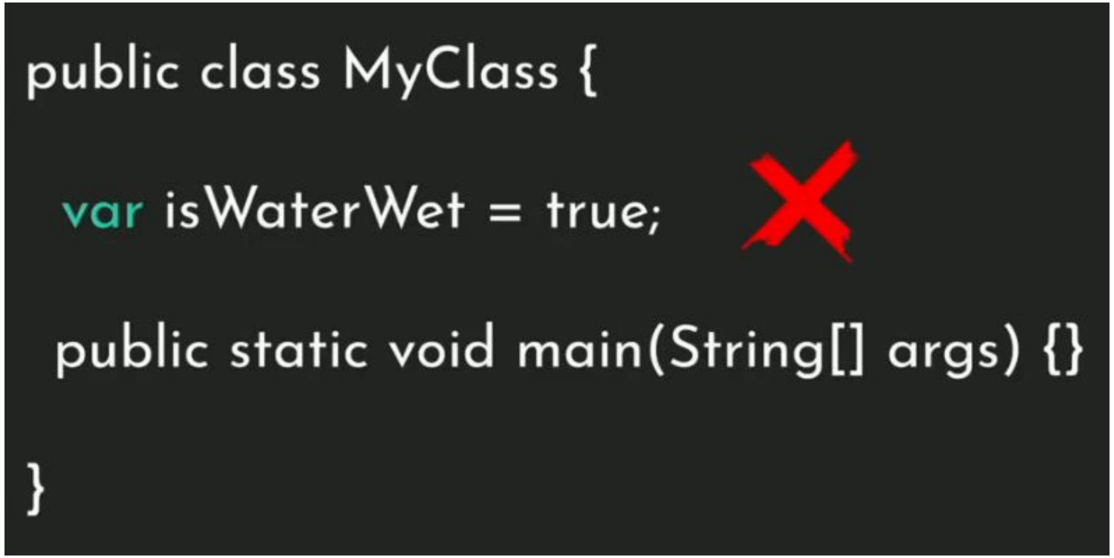
## Local Variable Type Inference

*When declaring a local variable as var, Java will infer its type.*



```
public static void main(String[] args) {
   var isWaterWet = true;
}
```

## Local Variable Type Inference



```
public class MyClass {

  var isWaterWet = true;

 public static void main(String[] args) {}

}
```

Questions ?