# Conceptual Modeling and Programming in GIScience

## Lecture 1: Introduction

Yingjing Huang
*yingjing.huang@univie.ac.at*

2026S – April 13, 2026

# What You Will Learn in This Lecture

## Part 1 – Course Overview

- What this course is about, how it works, and how you'll be graded

## Part 2 – Programming Fundamentals

- Understand what **programming** is and why we use **Java**
- Learning about **variables**, **data types**, and **operators**
- Understand **control structures** (if/else, for, while)

## Part 3 – Hands-On

- Set up and use **IntelliJ IDEA**
- First look at debugging and documentation (self-study)

# Course Objectives and Learning Outcomes

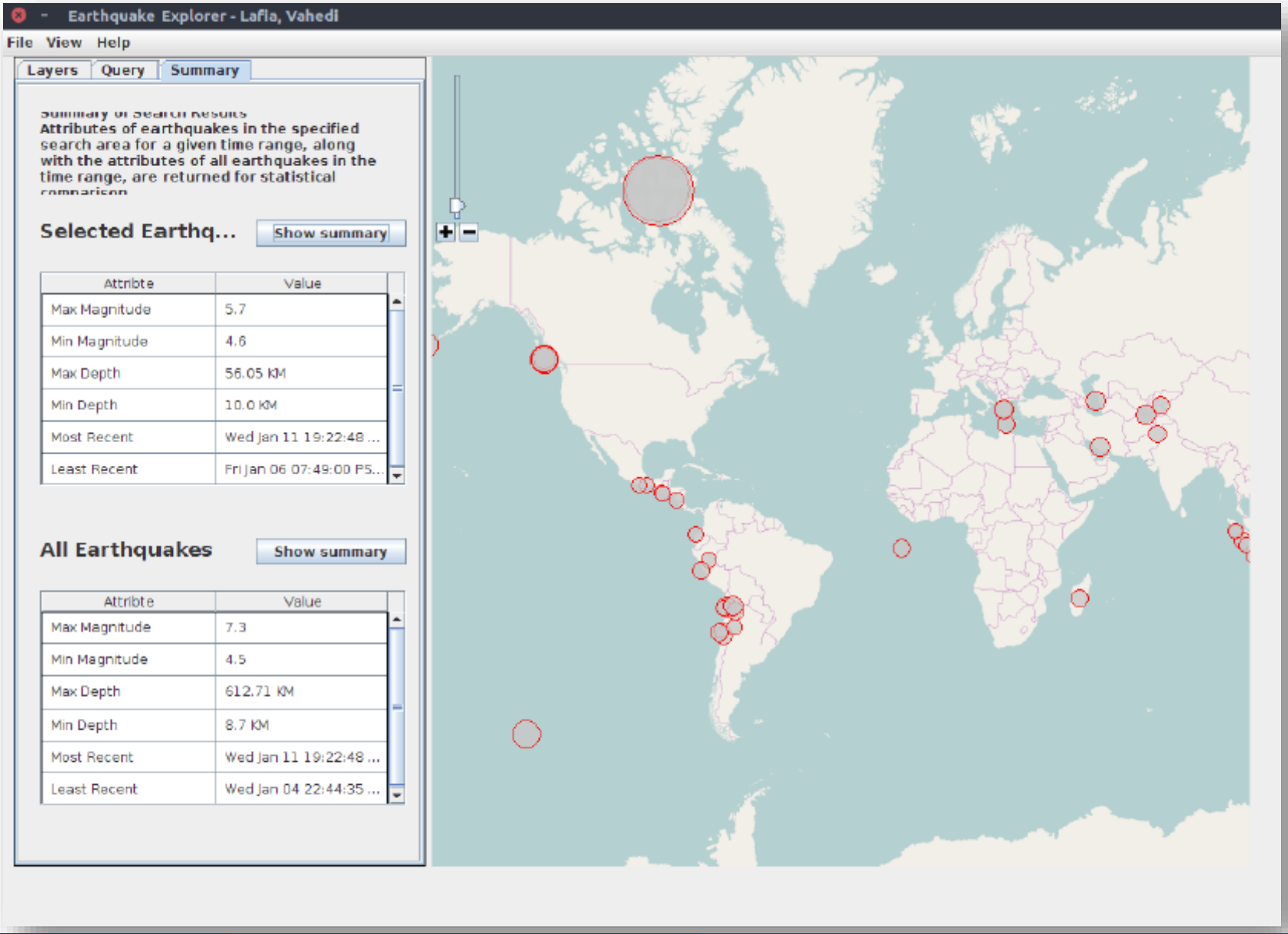
- Understand the challenges of **modeling real world** computationally
- Understand how GIS software works **under the hood**
- Learn the **basics** of **object-oriented programming**
- **Develop small GIS programs** and solve related problems
- Work in **teams**, communicate effectively and deliver under time pressure.

## You will learn how to:

- Implement **OGC Simple Feature** classes such as points, polylines, and polygons
- Implement basic **geostatistics**, e.g., nearest neighbor analysis
- Implement basic **algorithms** for geo-data, e.g., convex hull for home range estimation
- Implement a small **map on-screen digitizing** application
- Implement interactive **graphical user interfaces (GUIs)** and animations (e.g., a geo-game)
- **Read and write** files

**All this may be adjusted based on your interests and the progress we make**

# Examples From the Past



# A Few Things to Keep in Mind

- Read the schedule and **syllabus**

  (may/will change during the course)

  Available on **Moodle** and **GitHub**

- This is a course for **beginners**, but with a **steep learning curve**

  i.e.,a Master's level class

https://moodle.univie.ac.at/course/view.php?id=465688

https://yingjinghuang.github.io/290701-Conceptual-Modelling-and-Programming/

# Preliminary Schedule

| | |
|---|---|
| 13.04.2026 – W1 | Intro Data Types, Operators, and Control Statements |
| 20.04.2026 – W2 | Methods and Object Orientation I |
| 27.04.2026 – W3 | Methods and Object Orientation II |
| 04.05.2026 – W4 | Inheritance and Polymorphism I |
| 11.05.2026 – W5 | Inheritance and Polymorphism II |
| **18.05.2026 – W6** | **MIDTERM EXAM** |
| 01.06.2026 – W7 | GUI I and Action Listeners |
| **08.06.2026 – W8** | GUI, File Readers, and Model-View-Controller + **Complementary Assignment** |
| 15.06.2026 – W9 | Ripley's K Worked Example |
| 22.06.2026 – W10 | External Libraries and Exceptions |
| **29.06.2026 – W11** | **FINAL EXAM** |

Unfortunately, **we cannot offer make-up exams**, e.g., if you are traveling, working, etc.
Both the midterm and final exams require **writing source code on paper**.

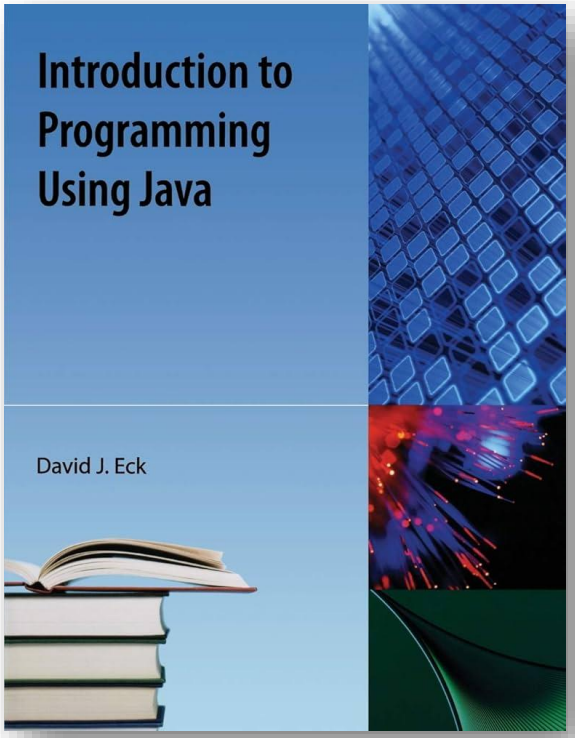
I cannot teach you programming

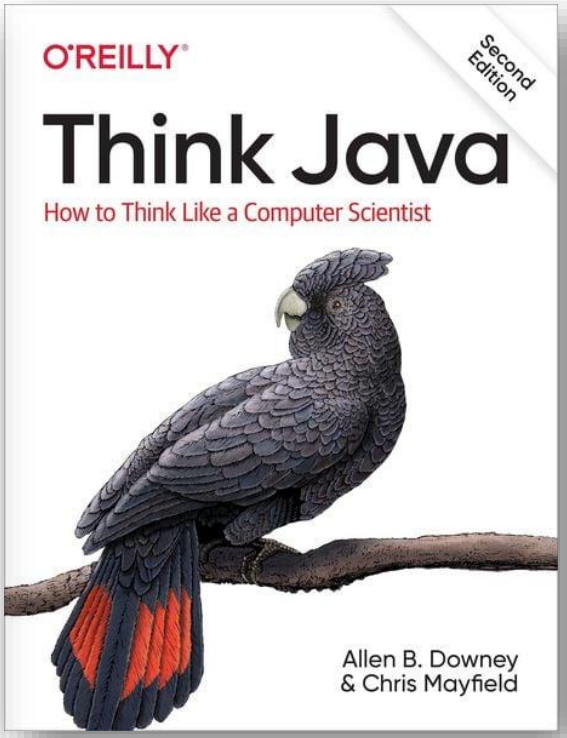
**You need to do it to learn how it works!**

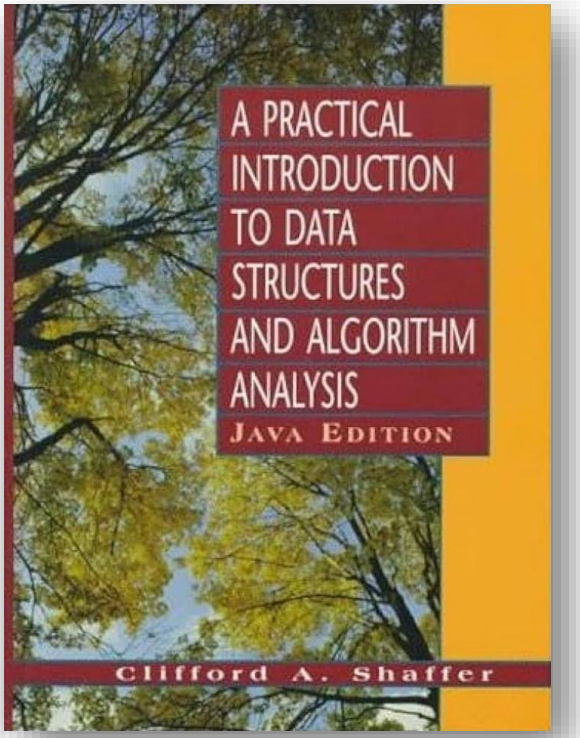
- Short theory; more **hands-on exercises**
- Learn by **example**: start with a problem, then figure out the concepts
- Work in **teams** and on small (about 5-10) **projects**

Many alternative textbooks,

but **please pick at least one**



# Bring Your Own Device is highly recommended

IDEs are a matter of personal preferences, use **IntelliJ IDEA** for as a fully-featured IDE


IntelliJ IDEA

# Write Your Own Code

- AI is good at coding — **but you only learn by doing it**
- One **mandatory assignment** (Week 8)
- Weekly optional assignments — **highly recommended**,

  they build on each other and prepare you for the exams

# Grading: Exams and Assignments

## 40% | Mid-Term Exam

**Date:** 18.05.2026
**Format:** 60+ min, writing source code *on paper*.

## 40% | Final Exam

**Date:** 29.06.2026
**Format:** 60+ min, writing source code *on paper*.

## 10% | Mandatory Quiz/Assignment

Most weekly assignments build up on each other, not assigning a grade does not imply that you should not do them.

## 10% | Active Participation

e.g., not just being present, asking questions, answering questions, explaining code,...

# Programming Basics

# What is Programming?

- Programming is the process (or art) of designing, **developing, debugging**, maintaining, and documenting **source code**
- Source code is a **collection** (not necessarily a sequence) of **instructions** formulated using a specific programming **language** that is translated to machine code and executed on a computer

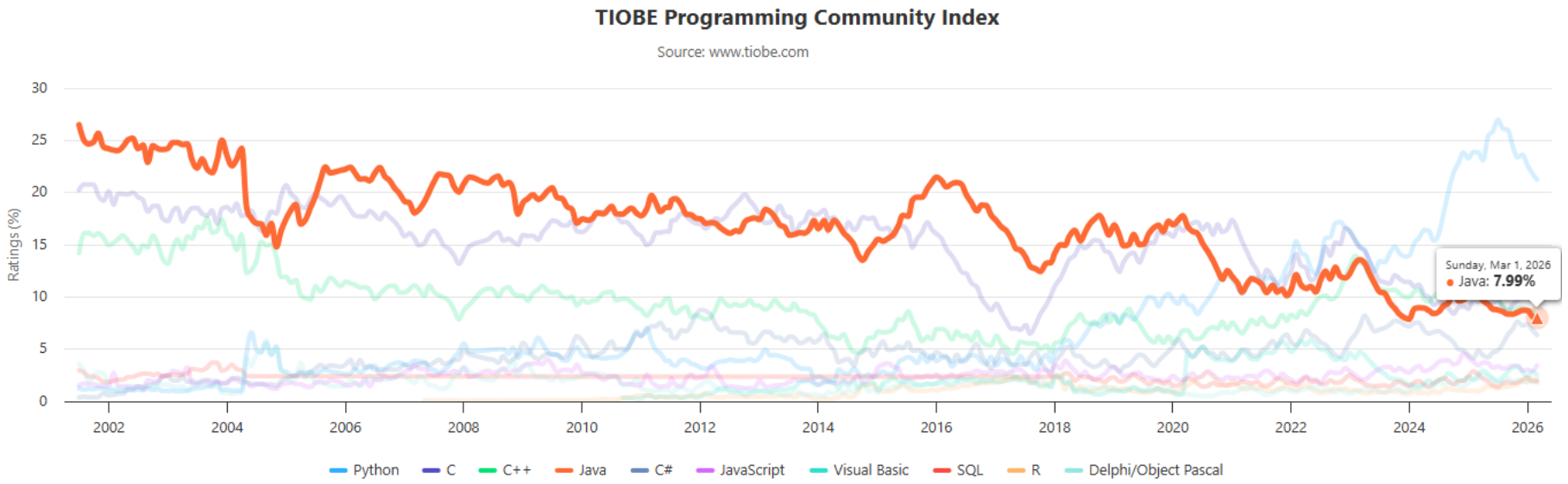
# Programming Languages

- A programming language provides the **syntax and semantics** to communicate instructions to a computer
- Many languages exist with different designs, but **most can implement the same functionality**
- No formal definition separates a "**script**" from a "**program**" — scripts are typically smaller, specific-purpose programs

# Java in a Nutshell

- A **general-purpose, object-oriented** programming language
- **"write once, run anywhere"** – runs on all major platforms (clients, servers, mobile, game consoles, etc.)
- One of the **most popular** programming languages, with 9+ million active developers (Oracle)
- Source code is **compiled** to **bytecode** that runs on the **Java Virtual Machine** (JVM), independent of computer architecture

# Why Java?




- Java runs on **all major server**, desktop, and mobile platforms
- Java's syntax is very close to C, C++, and C# — **learning one gives you access to the whole family**
- Java was designed from the ground up for **object-oriented programming**

# Variables

- A variable in Java has a **type**, a **name**, and a **value**
- The value is **variable** - it can be changed
- Java is a strongly **typed**: every variable must have a declared type

```
// Declare and initialize
int age = 25;
double latitude = 48.2082;
boolean isCapital = true;

// The value can be changed
age = 26;
latitude = 47.0707;
```

# Primitive Data Types (Overview)

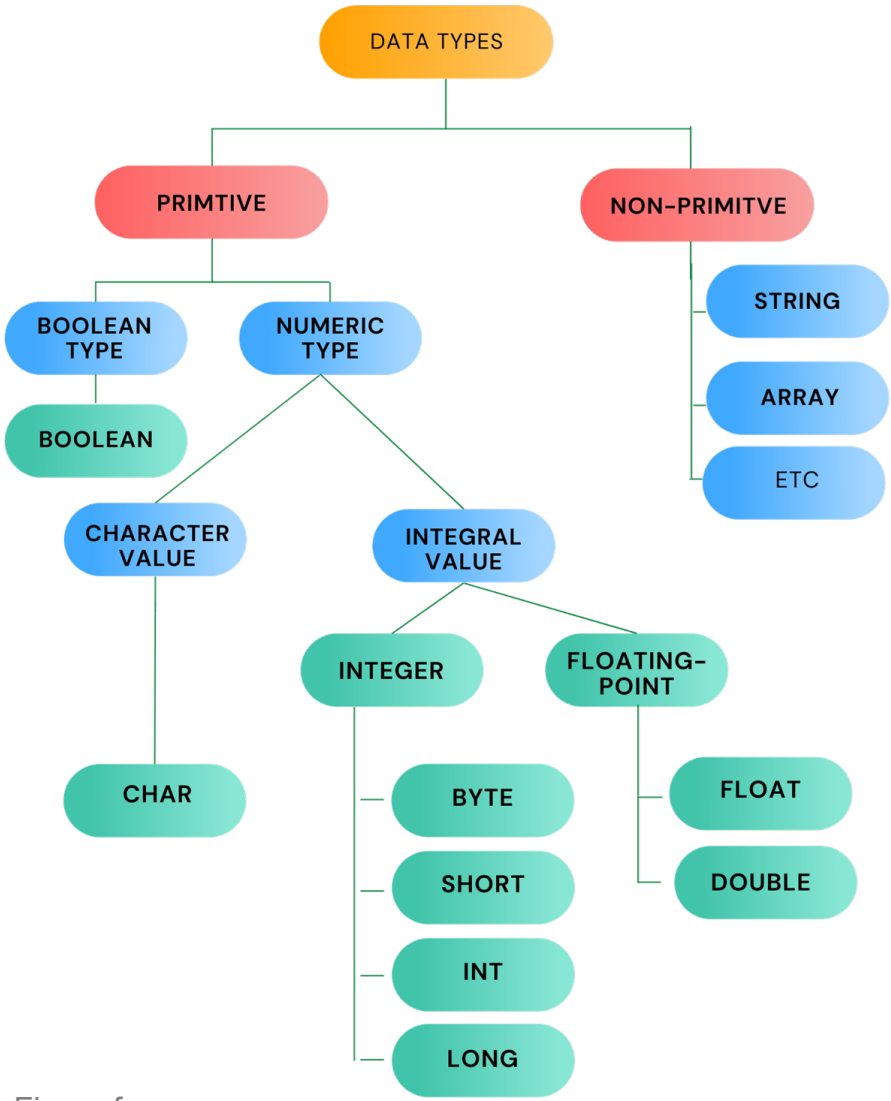



Figure from:
https://logicmojo.com/data-types-in-java

- 8 primitive types, 4 categories
- The most commonly used:

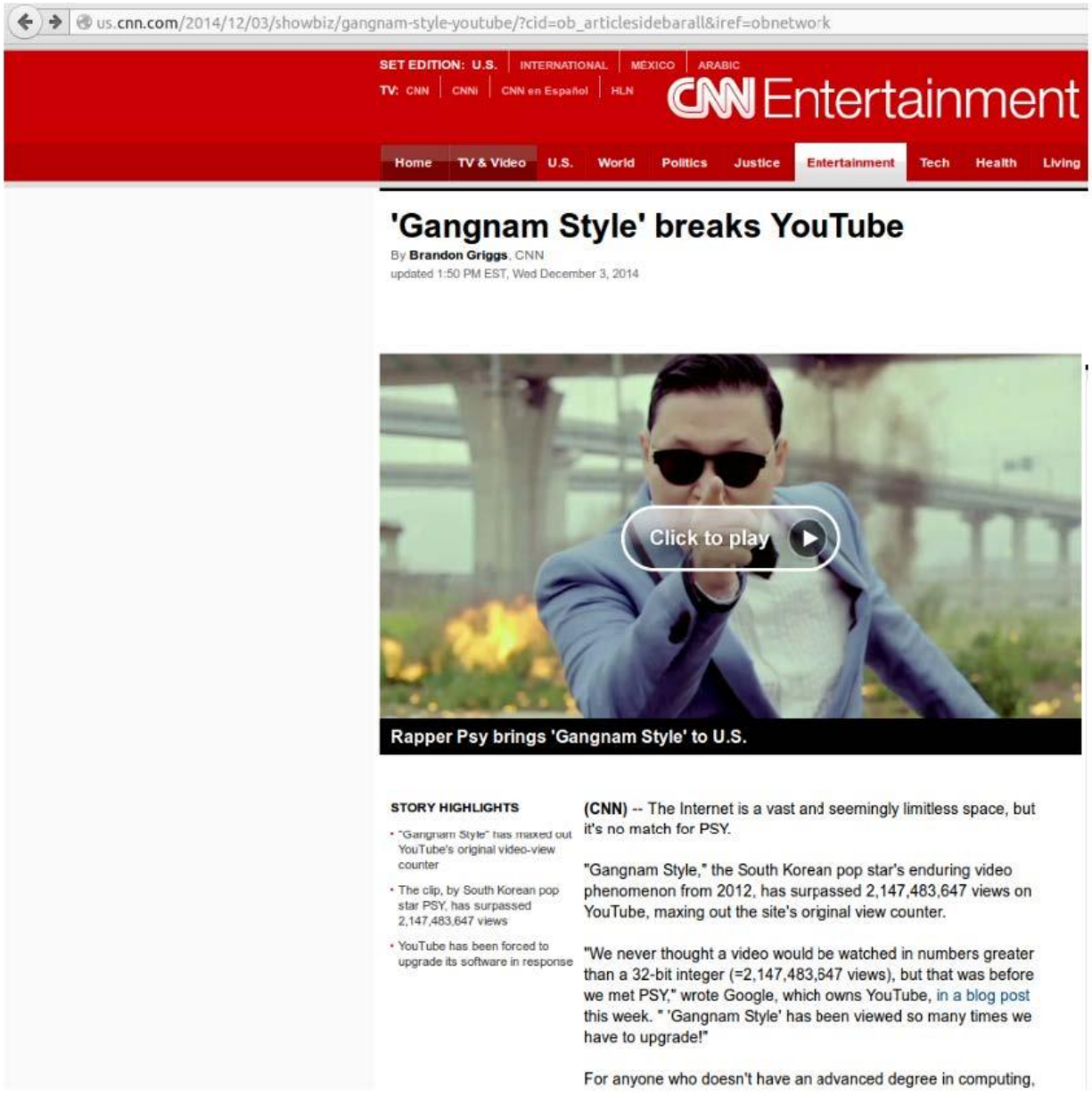
  ***int*, *double*, *boolean***

# Integer Types

| Type | Size (bits) | Min. value | Max. value |
|---|---|---|---|
| byte | 8 | -128 | 127 |
| short | 16 | -32,768 | 32,767 |
| int | 32 | -2,147,483,648 | 2,147,483,647 |
| long | 64 | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |

In practice, use ***int*** for most cases. Use ***long*** when int is not enough.

# Why Should You Care?



us.cnn.com/2014/12/03/showbiz/gangnam-style-youtube/?cid=ob_articlesidebarall&iref=obnetwork

SET EDITION: U.S. | INTERNATIONAL | MÉXICO | ARABIC

TV: CNN | CNNi | CNN en Español | HLN

CNN Entertainment

Home | TV & Video | U.S. | World | Politics | Justice | Entertainment | Tech | Health | Living

## 'Gangnam Style' breaks YouTube

By **Brandon Griggs**, CNN

updated 1:50 PM EST, Wed December 3, 2014



Rapper Psy brings 'Gangnam Style' to U.S.

**STORY HIGHLIGHTS**

- "Gangnam Style" has maxed out YouTube's original video-view counter
- The clip, by South Korean pop star PSY, has surpassed 2,147,483,647 views
- YouTube has been forced to upgrade its software in response

**(CNN)** -- The Internet is a vast and seemingly limitless space, but it's no match for PSY.

"Gangnam Style," the South Korean pop star's enduring video phenomenon from 2012, has surpassed 2,147,483,647 views on YouTube, maxing out the site's original view counter.

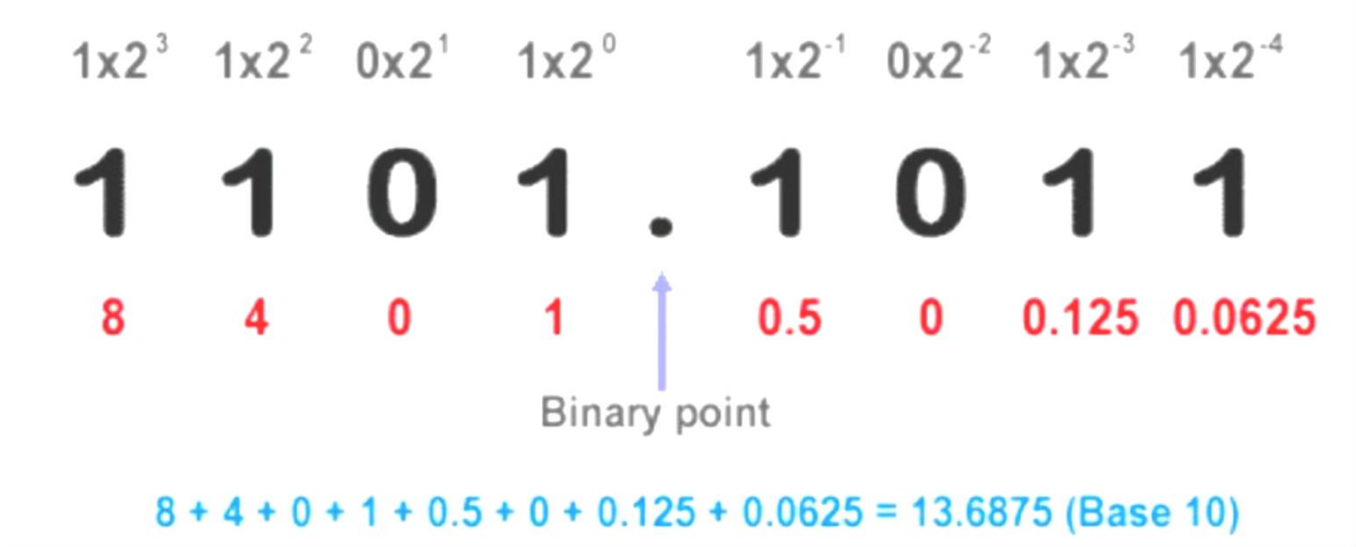
"We never thought a video would be watched in numbers greater than a 32-bit integer (=2,147,483,647 views), but that was before we met PSY," wrote Google, which owns YouTube, in a blog post this week. " 'Gangnam Style' has been viewed so many times we have to upgrade!"

For anyone who doesn't have an advanced degree in computing,

# Decimal Types

| Type | Size (bits) | Precision | Range |
|---|---|---|---|
| float | 32 | ~6-7 digits | $\approx \pm 3.4e^{38}$ |
| double | 64 | ~15 digits | $\approx \pm 1.7e^{308}$ |

- Use ***double*** by default – Java treats decimal literals (e.g., 3.14) as double
- Do not use ***double*** for precise computations (e.g., currencies). Use ***BigDecimal*** instead.




- **Computers store everything in binary** (0s and 1s)
- Left of the dot: **powers** of 2 (8, 4, 2, 1, …)
- Right of the dot: **fractions** (1/2, 1/4, 1/8, …)

# Decimal Numbers in Binary

- Some decimals work perfectly:

  13.6875 = 1101.1011

- Others don't:

  0.1 = 0.000110011… (repeats forever)

- Doubles in Java are **approximations** – accuracy vs. storage
- For exact precision (e.g., currencies), use ***BigDecimal***

## *char*

- ***char*** stores a single Unicode character (2 bytes)
- Uses single quotes: **'A', '7', '@'**
- **'7' (*char*) ≠ 7 (*int*)**

## *boolean*

- Only two values: ***true*** or ***false***
- Used in every ***if***, ***while***, and loop condition

**Type casting**:
converting a value from one data type to another

## Widening

- smaller type → larger type, done automatically
- **byte → short → char → int → long → float → double**
- Example:

```
int i = 123;
long l = i;  // ✓ widening, automatic
```

## Narrowing

- Larger type → smaller type, **must cast explicitly**
- Syntax: **(targetType) value**
- Example:

```
double d = 123.99;
int i = (int) d;  // i = 123, decimal part lost
```

# Operators - Overview

| | |
|---|---|
| **Arithmetic Operators** | + - * / % |
| **Assignment Operators** | = += -= *= /= %= |
| **Unary Operator** | + - ++ -- |
| **Comparison Operators** | == != > < >= <= |
| **Logical Operators** | && \|\| ! |
| **Bitwise Operators** | & \| ^ >> << >>> ~ |

| | |
|---|---|
| **Arithmetic Operators** | + - * / % |
| **Assignment Operators** | = += -= *= /= %= |
| **Unary Operator** | + - ++ -- |

```
int a = 10 % 3;    // a = 1
int x = 5;
x += 3;            // x = 8
x++;               // x = 9
```

# Operators – Comparison, Logical & Bitwise

| | |
|---|---|
| **Comparison Operators** | == != > < >= <= |
| **Logical Operators** | && \|\| ! |
| **Bitwise Operators** | & \| ^ >> << >>> ~ |

```
int x = 10;
x == 10    // true
x > 5 && x < 20   // true
```

- **=** is assignment, **==** is comparison
- **&&** (logical) vs **&** (bitwise) — don't confuse them

# Java Keywords

- These 61 keywords together with the syntax of **operators** and **separators** define the Java programming language
- **Keywords cannot be used as variable names**

| | | | | | |
|---|---|---|---|---|---|
| abstract | assert | boolean | break | byte | case |
| catch | char | class | const | continue | default |
| do | double | else | enum | exports | extends |
| final | finally | float | for | goto | if |
| implements | import | instanceof | int | interface | long |
| module | native | new | open | opens | package |
| private | protected | provides | public | requires | return |
| short | static | strictfp | super | switch | synchronized |
| this | throw | throws | to | transient | transitive |
| try | uses | void | volatile | while | with |
| _ | | | | | |

**Table 1-1** The Java Keywords

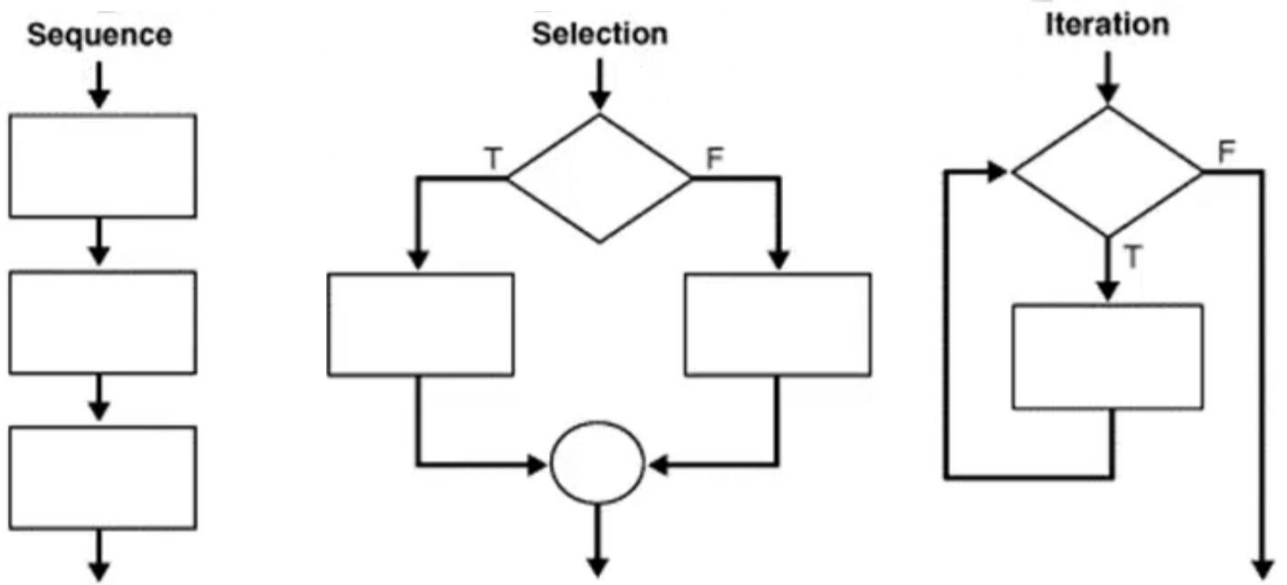
# Summing Up

- **Declare:** specify the type and name
- **Initialize:** assign a value
- **Operate**: use operators to compute and assign new values

```
int number;         // 1. Declare
number = 5;         // 2. Initialize
number = 5 + 6;     // 3. Operate
```

- **Sequence**: code runs line by line, top to bottom
- **Selection**: choose a path based on a condition (**if**/**else**)
- **Iteration**: repeat a block of code (**for**, while)

Every program is built from these three building blocks.



## Selection

- ***if (condition) { } else if { } else { }***
- ***switch*** — compare one value against many options

## Iteration

- ***for (init; condition; step) { }***
- ***while (condition) { }***
- ***break*** — exit a loop early

```java
// if-else
if (temp > 40) {
    System.out.println("Anomaly!");
} else {
    System.out.println("Normal");
}

// for
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}

// while
int count = 0;
while (count < 5) {
    count++;
}
```

- A **reusable** block of code that performs a specific task
- Java has many **built-in methods**

  e.g., *System.out.println("Hello World!")*
- Programming is largely about using methods defined by

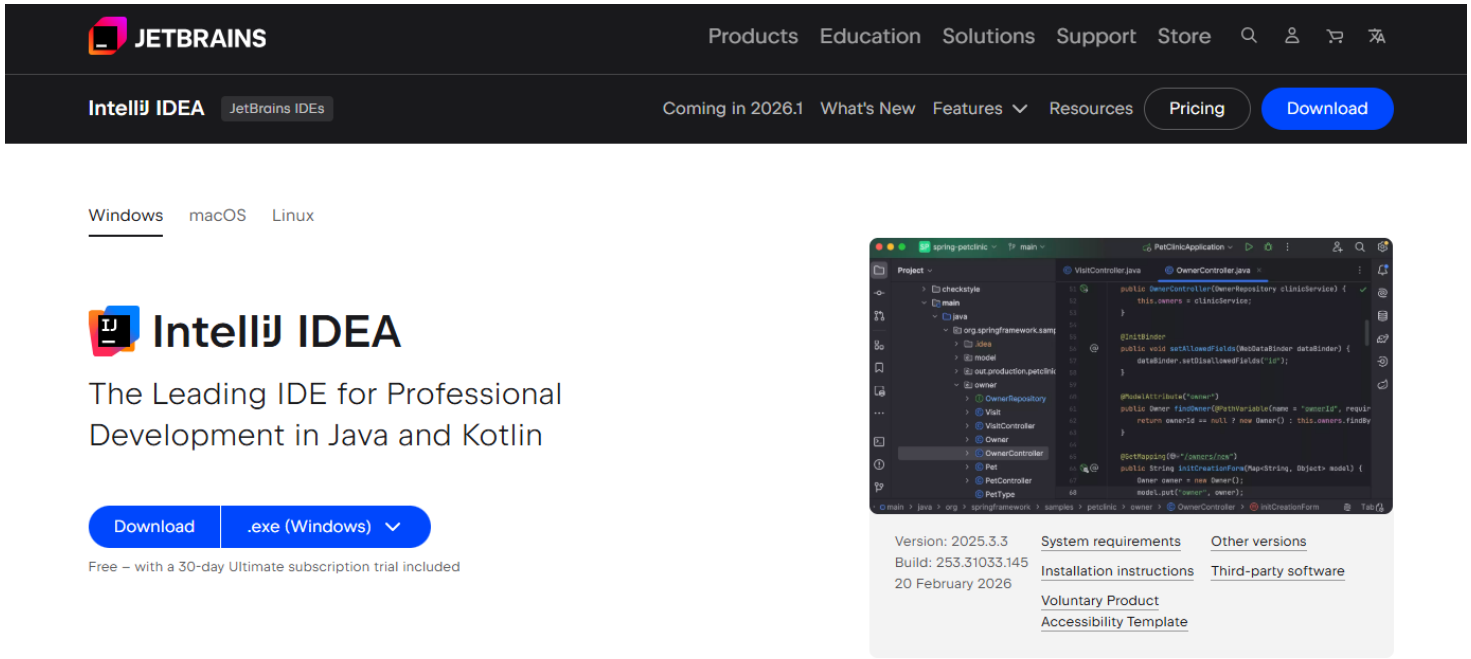
  **Java**, **others**, and **yourself**
- Code runs **line-by-line** within a method, but not

  necessarily across a whole program

# Hands-On

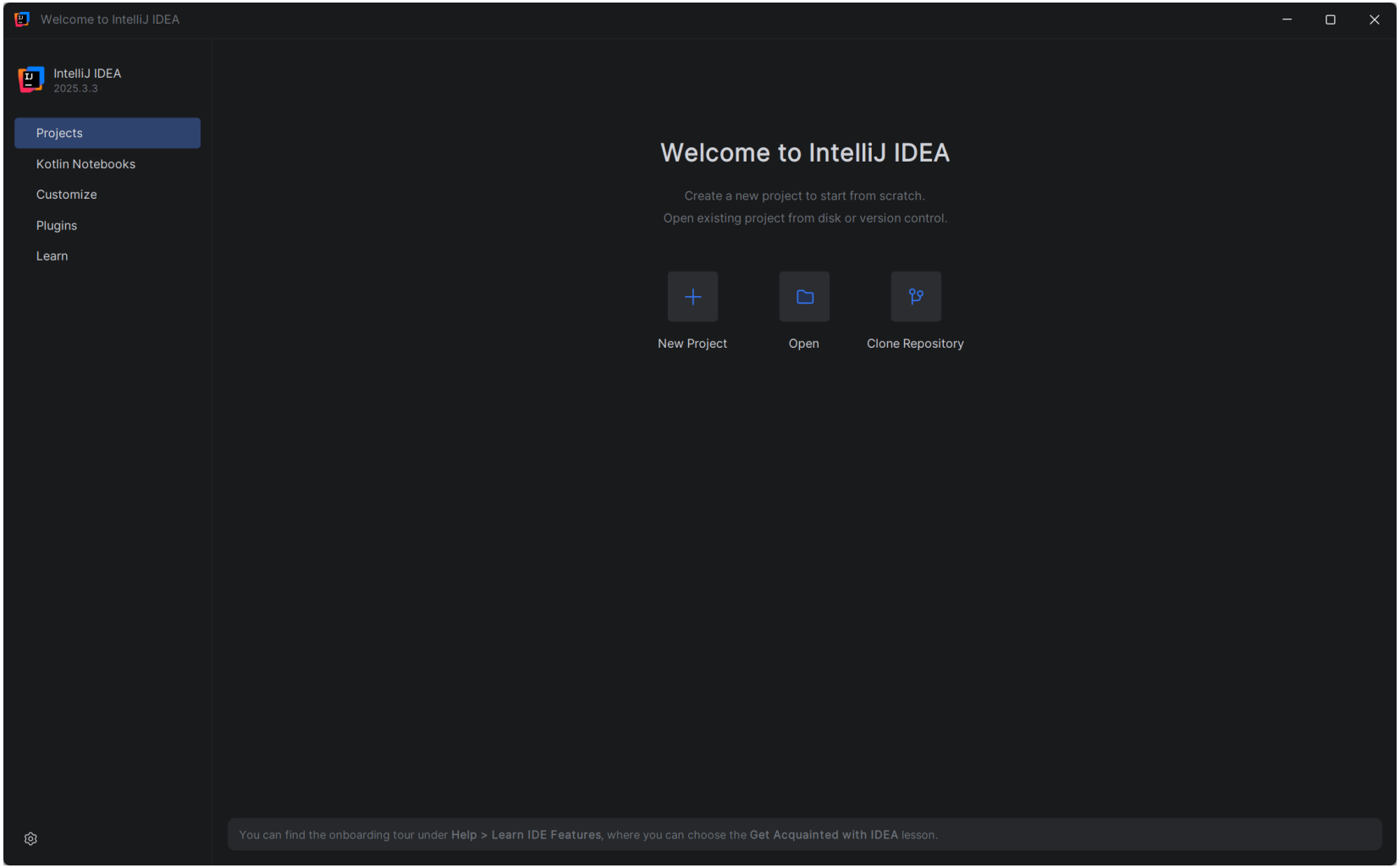
# Tools and Practice

# The IntelliJ IDEA IDE in a Nutshell

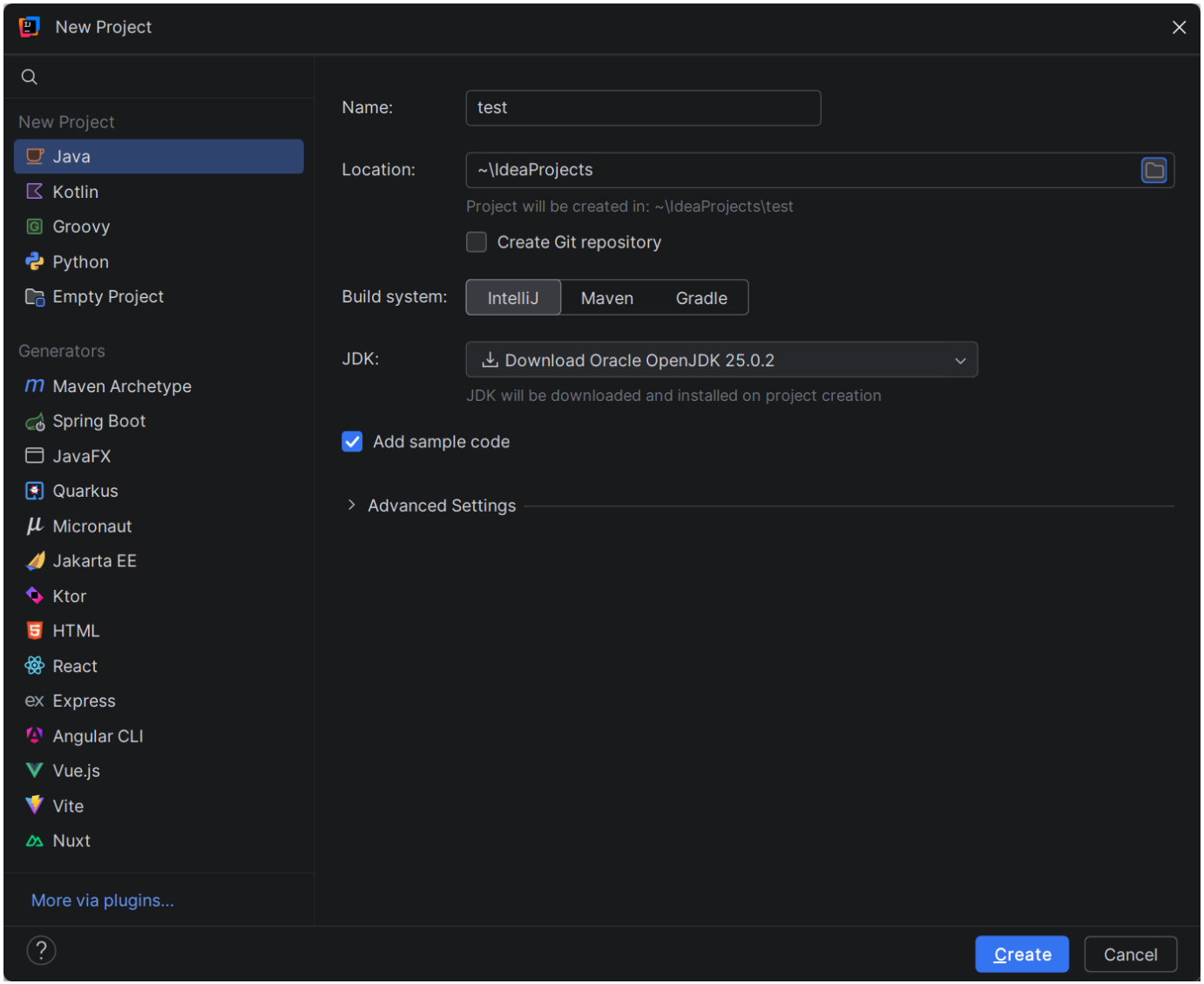



https://www.jetbrains.com/idea/download/?section=windows

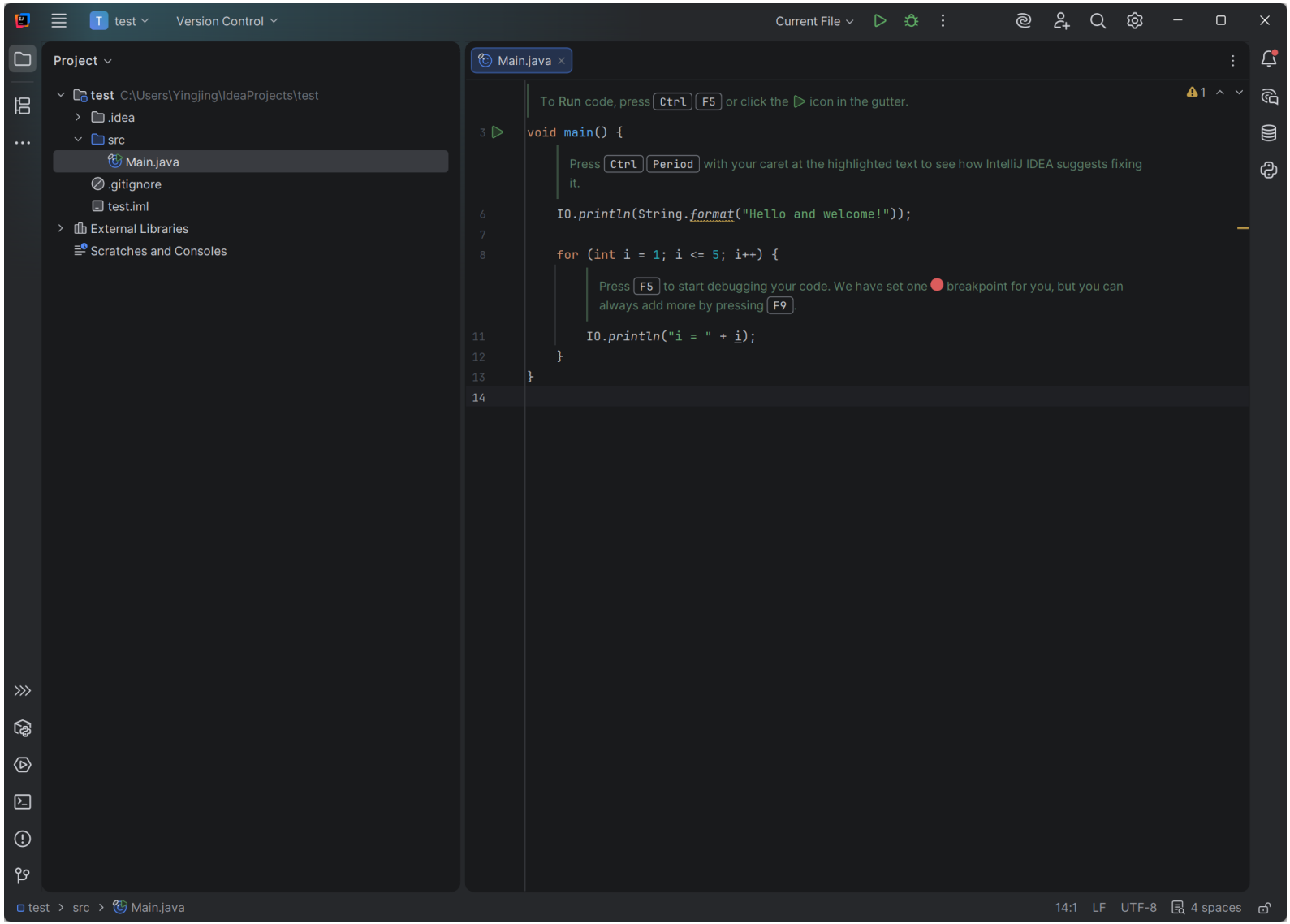
# The IntelliJ IDEA IDE in a Nutshell



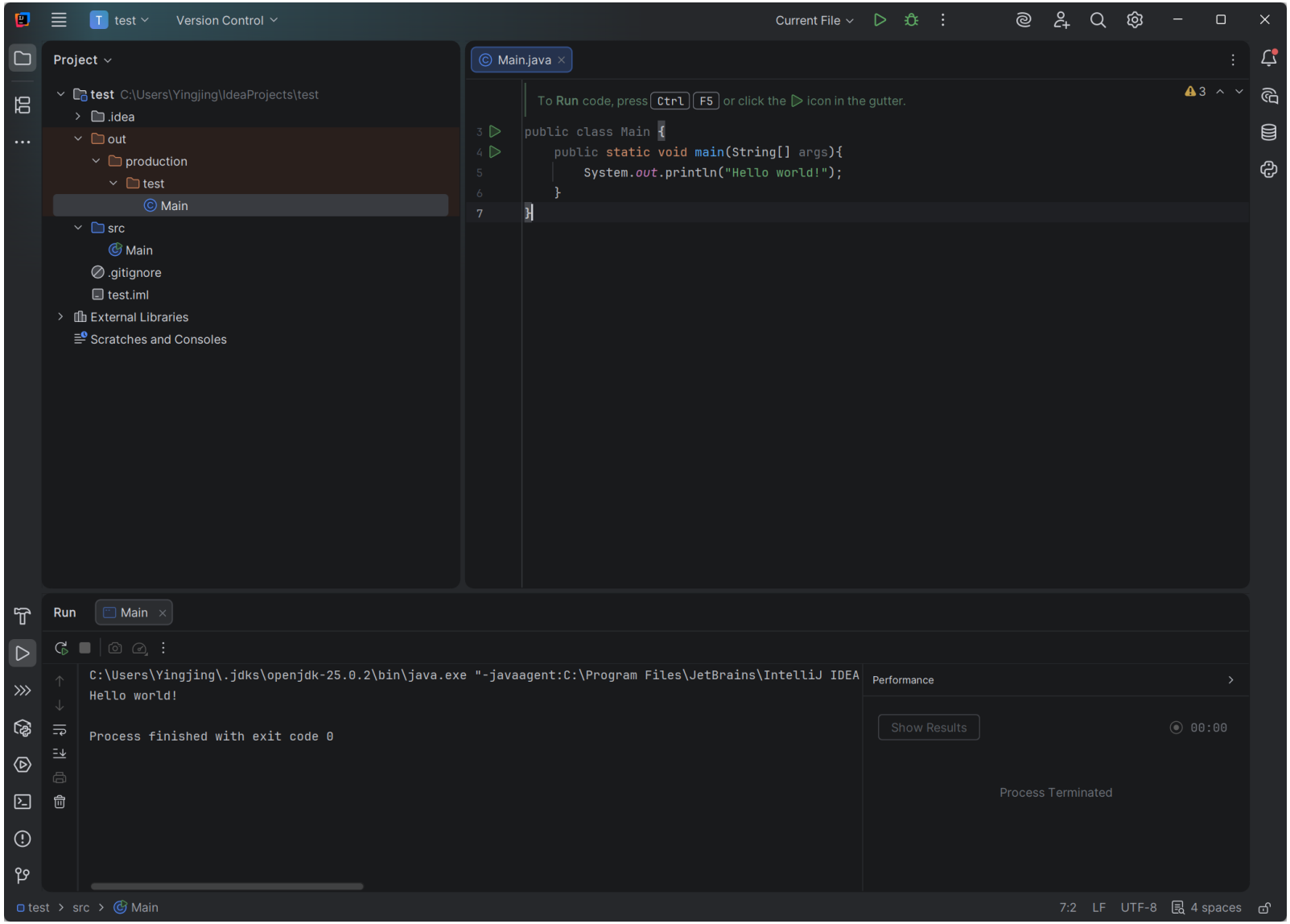
# The IntelliJ IDEA IDE in a Nutshell



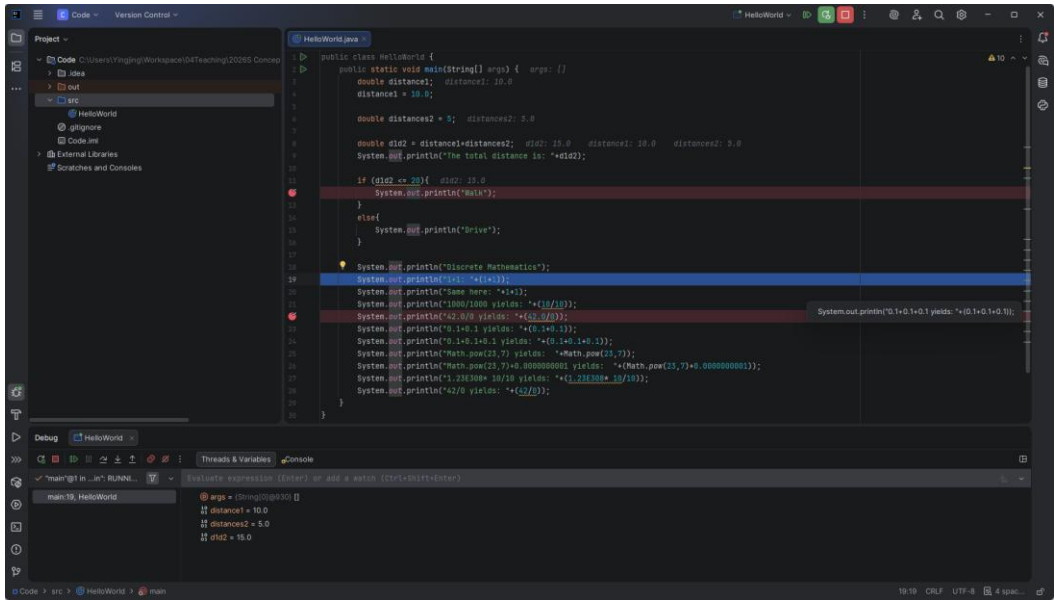
# The IntelliJ IDEA IDE in a Nutshell



# The IntelliJ IDEA IDE in a Nutshell



- **Debugging:** finding and resolving defects in your code
- **Breakpoint:** pause execution at a specific line
- **Watch & Inspect:** examine variable values at runtime



## IntelliJ IDEA

Keyboard Shortcut Cheat Sheet

### Navigating the Editor

| | |
|---|---|
| Ctrl+N | Find a class |
| Ctrl+Shift+N | Find a file |
| Ctrl+Alt+←→ | Navigate forward |
| Ctrl+E | Recent files |
| Alt+F7 | Alt usages |

### Editing Code

| | |
|---|---|
| Ctrl+Alt+L | Reformat code |
| Ctrl+Alt+O | Optimize imports |
| Ctrl+D | Duplicate current line or selecton |
| Ctrl+Y | Delete current line |
| Ctrl+Shift+↑ | Move line up/down |

### Code Generation

| | |
|---|---|
| Alt+Insert | Generate code, (constructor, getter, setter, etc.) |
| Ctrl+O | Override methods |
| Ctrl+I | Implement methods |

### Searching

| | |
|---|---|
| Ctrl+Shift F | Find in path |
| Ctrl+F | Find in current file |
| Ctrl+R | Replace in current file |

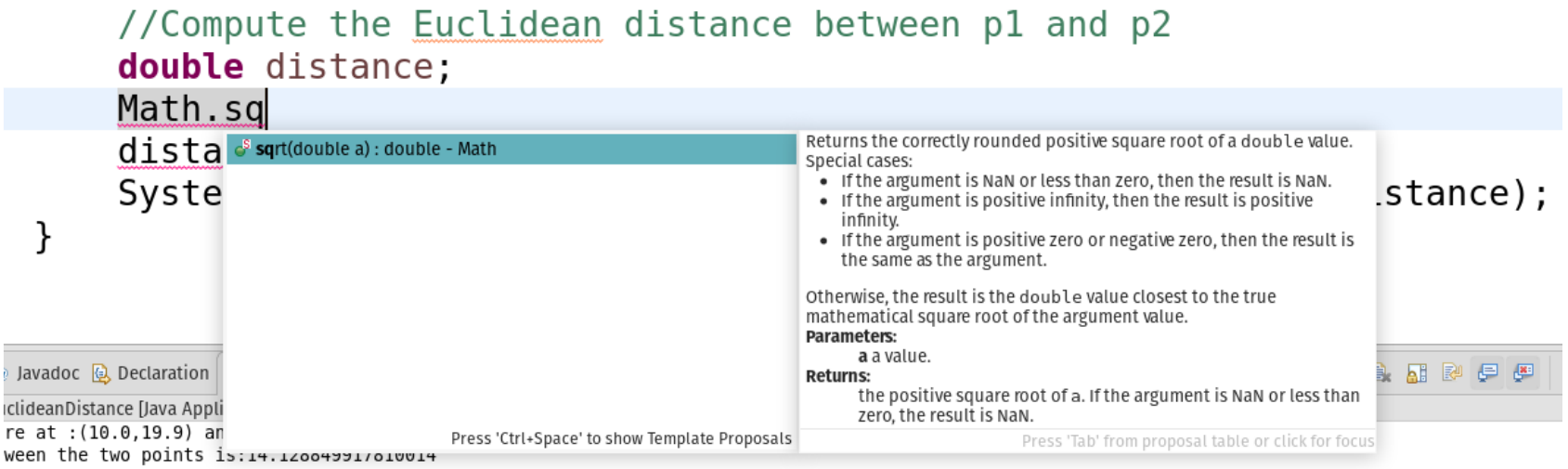
### Running and Debuggi

| | |
|---|---|
| Shift+F10 | Run |
| Shift+F9 | Debug |
| F8 | Step over |
| F7 | Step into |

### Other Useful Shortcuts

| | |
|---|---|
| Ctrl+Space | Basic code completion |
| Ctrl+Shift+Space | Smart code completion |
| ↑ //(double tap) | Search everywhere |
| Ctrl+P | Show method parameter info |

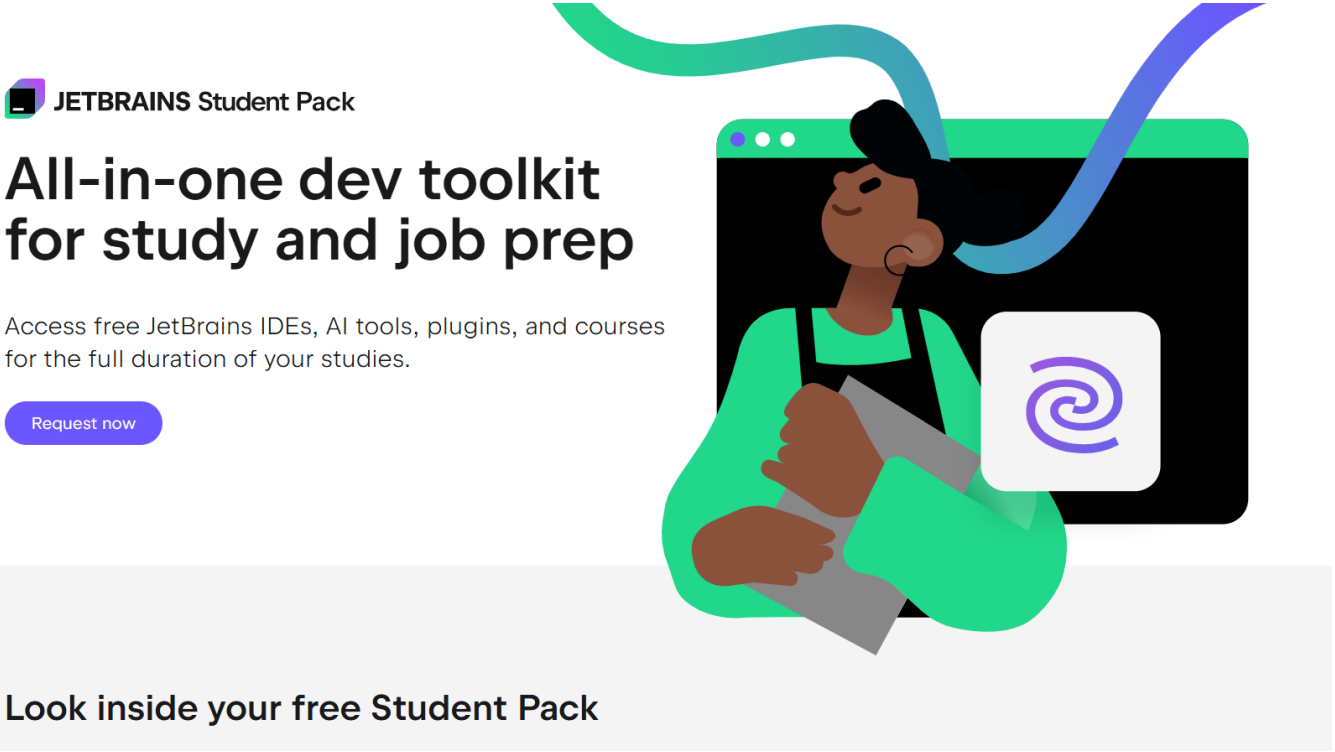
# Documenting your Code




- Without documentation, you wouldn't know how to use Java's built-in classes — or anyone else's code
- After a few weeks, you may not even understand your own code
- Java has a built-in tool called **JavaDoc** to help you create documentation

You can explore more on your own….

And don’t forget to obtain your student pack




https://www.jetbrains.com/academy/student-pack/

## Reading

- Read about **Variables & Control Statements (a typical chapter 1+2)** in your favorite textbook.
- Be able to **describe** what a **for loop** and **if statement** is in your own words.

## Coding

- Compute the **Euclidean** distance between two **points** in a **2D Cartesian** coordinate system (hint: ***Math.sqrt()*** and ***Math.pow()***)
- Define a **polyline** and a **polygon** and compute their shortest **distance between both using their vertices** (hint: you can use separate variables for each point)
- **Optional**: check whether the polygon is **closed**.

## Submission

- Upload a zip file **[FirstNameLastNameW1.zip]** with the *.java files
- **Deadline**: Sundays at 5:00 PM (Vienna Time) (The day before our Monday lectures).

# Your Turn: Exercises

```java
package at.ac.univie.gis.week1;

/**
 * Week 1 - Introduction to Java basics: output, arithmetic, variables, loops, and methods.
 */
public class Week1 {

    public static void main(String[] args) {

        // --- Section 1: Output and basic arithmetic ---
        System.out.println("Discrete Mathematics");

        // Parentheses force arithmetic addition: 1+1 = 2
        System.out.println("1+1: " + (1 + 1));

        // Without parentheses, + performs string concatenation: "Same here: " + "1" + "1" = "Same here: 11"
        System.out.println("Same here: " + 1 + 1);

        // Integer division: 10/10 = 1 (no decimal places)
        System.out.println("1000/1000 yields: " + (10 / 10));

        // WARNING: Integer division by zero throws ArithmeticException and crashes the program!
        // Uncomment the next line to see it in action (the program will stop here):
        // System.out.println("42/0 yields: " + (42 / 0));

        // Floating-point division by zero does NOT crash - it returns Infinity instead.
        // This is because IEEE 754 floating-point standard defines division by zero as Infinity.
        System.out.println("42.0/0 yields: " + (42.0 / 0));

        // Floating-point precision: 0.1 + 0.1 = 0.2 (exact)
        System.out.println("0.1+0.1 yields: " + (0.1 + 0.1));

        // But 0.1 + 0.1 + 0.1 is NOT exactly 0.3 due to floating-point representation!
        System.out.println("0.1+0.1+0.1 yields: " + (0.1 + 0.1 + 0.1));

        // --- Section 2: Variables ---
        double distance1;       // Declare a variable (no initial value yet)
        distance1 = 10;         // Assign a value later
        double distance2 = 5;   // Declare and initialize in one step

        // Math.max returns the larger of two values
        System.out.println("Max: " + Math.max(distance1, distance2));

        // Calling a custom method (defined below)
        System.out.println("add-one: " + addOne(distance2));

        // --- Section 3: Loops and Math functions ---
        // Print the square of each number from 0 to 9 using Math.pow()
        for (int i = 0; i < 10; i++) {
            System.out.println(Math.pow(i, 2));
        }

        // --- Section 4: Conditionals (uncomment to try) ---
        // int count = 4;
        // if (count == 42) {
        //     System.out.println("THE answer!");
        // } else {
        //     System.out.println("hmmm...");
        // }
    }

    /**
     * A simple method that adds 1 to the input value.
     * Demonstrates how to define and call a static method.
     */
    public static double addOne(double in) {
        return in + 1;
    }
}
```