Introduction OO and Java

Lecture 1 (26 January 2021)

Today

- Organization
- Object-orientation – an overview
- Java – an overview

Organization (1)

- Teachers
  - Pol van Aubel (first 3 weeks)
  - Sjaak Smetsers
- Practicum coordinator
  - Liye Guo (PhD student)
- Student assistants
  - …

Organization(2)

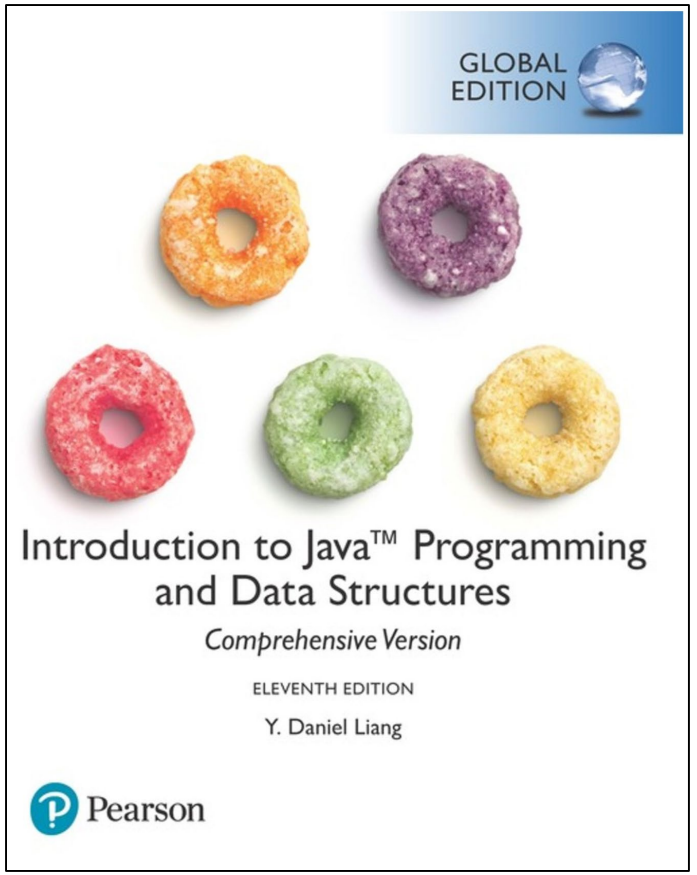
- Lecture
  - Tuesday morning 8:30 – 10:15
- Tutorial
  - Wednesday morning 8:30 – 10:15
- Computer lab (At Home)
  - Thursday afternoon (AI), Friday whole day (CS)

Computer lab

- Weekly assignments
- Regulation: **not a single FAIL!**
- FAIL if
  - Incomplete program: program does not compile.
  - Essential parts are missing
- Deadlines are strict!
- At most 2 assignments can be resubmitted.
- **If you don’t comply with the regulation, you will be excluded from the exam + resit.**
- Done OO before? You may reuse previous solutions.

Organization (3)

- Book (recommended, not compulsory):
  - Intro to Java Programming, Comprehensive Version
  - Liang



Learning goals

- introduce enumeration types as classes.
- use interfaces and inheritance.
- use and extend/adjust given classes.
- use generics.
- understand and apply collections (like lists and the associated iterators).
- use streams to perform computations upon elements of collections.
- apply software design patterns.
- develop and use JUnit tests.
- use object-oriented GUI libraries (JavaFX) and event driven programming.
- divide the work over threads, synchronize threads, handle race-conditions, avoid deadlock.

Object-orientation

- Design and programming in OO-style
- OO Design: from requirements to specification
  - Requirements Engineering (NWI-IPC023)
- OO Programming: from specification to implementation
  - This course

(Software) Objects

- Building blocks of software systems
  - a program is a collection of interacting objects
  - objects cooperate to complete a task
  - to do this, they communicate by calling (or invoking) each other's methods

Software Objects (2)

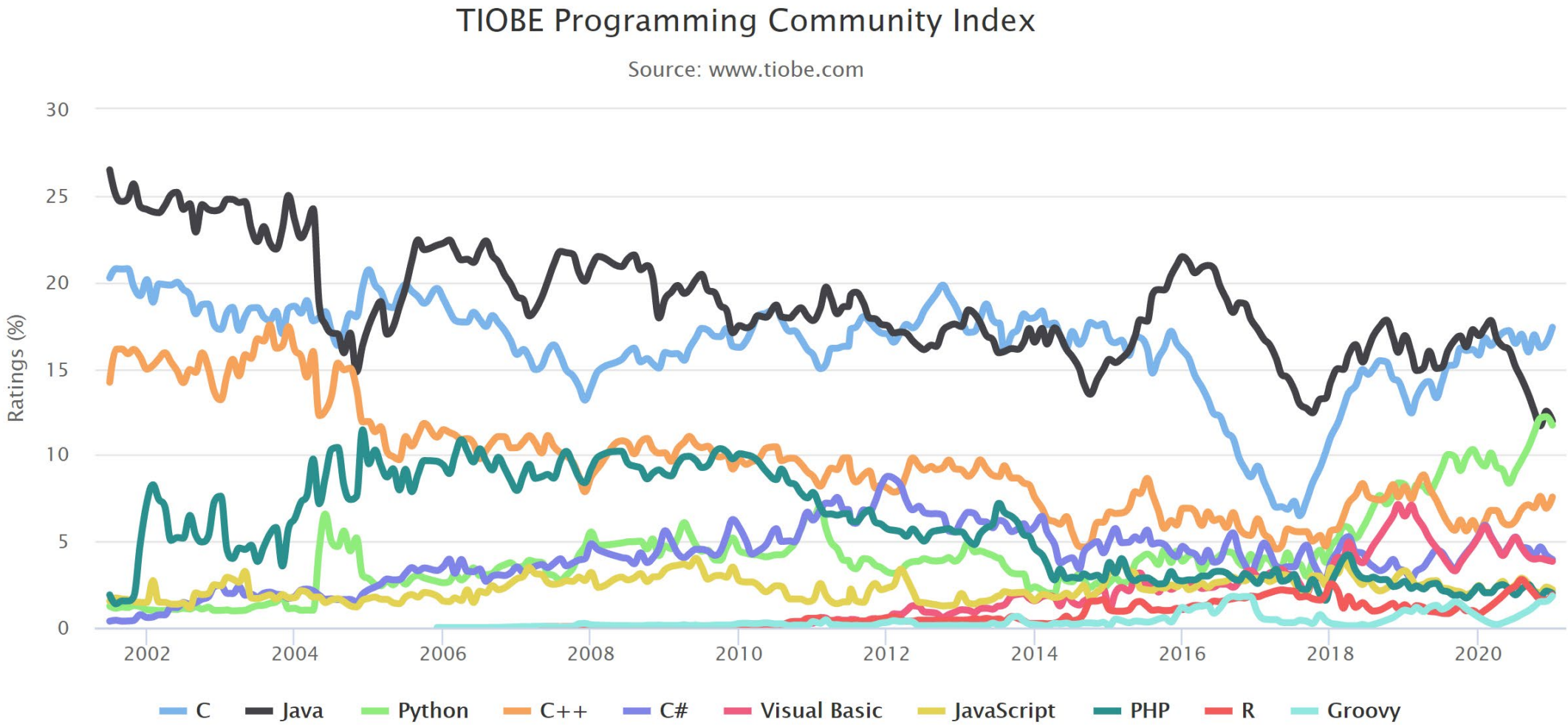
- Objects model tangible / concrete things
  - school, car, dog
- Objects model conceptual / abstract things
  - meeting, date, vehicle
- Objects model processes / tasks
  - finding a path through a maze, sorting a deck of cards, handling I/O

- Objects have responsibilities
  - capabilities: what they can do, how they behave
  - properties: features that describe them

OO language

- Object-oriented language
  - Programming in object-oriented languages is called *object-oriented programming* (*OOP*)
  - In this course: **Java**

Programming language ranking



THE PROBLEM ABOUT BEING A PROGRAMMER

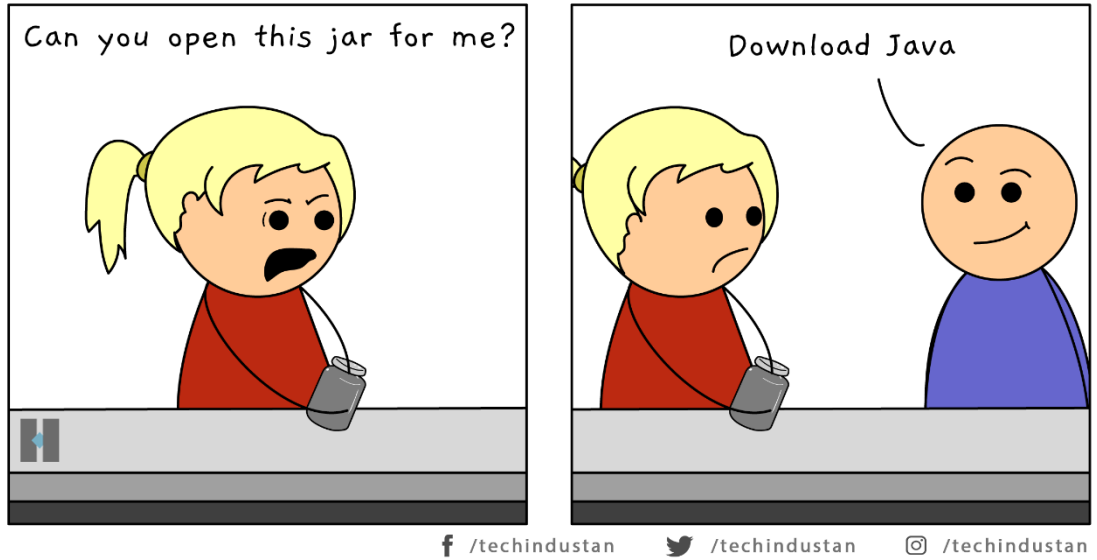
My mom said:
"Honey, please go to the market and buy 1 bottle of milk. If they have eggs, bring 6"

I came back with 6 bottles of milk.

She said: "Why the hell did you buy 6 bottles of milk?"

I said: "BECAUSE THEY HAD EGGS!!!!"

Java™



Imperative vs Object Oriented Programming

- Emphasis on algorithms
  - Top-down programming as a decomposition method
  - describes in detail the steps that the computer must take to accomplish the goal
- Program and data separated

- Emphasis on Software Systems
  - representing problems using real-world objects and their behaviour
  - describes how objects collaborate to accomplish the goal
- Data and code are bundled together

Classes and Instances

- Our ‘current conception’: each object corresponds directly to a *particular* real-life object, e.g., a specific atom or automobile
- How do we define them?
  - each object separately?
- Disadvantage: it’s much too impractical to work with objects this way
  - there may be arbitrarily many objects (e.g., modeling all stars in the galaxy)
  - some of them may have much in common

Classes and Instances (2)

- Objects are created from classes.
  - The class describes the kind of object (behaviour + properties) .
  - The objects represent individual instantiations of the class.
- Classifying objects factors out commonality among sets of similar objects
  - describe what is common just once (class)
  - then “stamp out” any number of copies later (instances)

Class example: Car

properties: instance variables/attributes

```
public class Car {
    private String brand;
    private int fuelConsumption;
    private String registrationNumber;
    private String color;

    public Car( String brand, int consumption, String color ) {
        this.brand = brand;
        this.fuelConsumption = consumption;
        this.color = color;
    }

    public void changeColor( String new_color ) {
        this.color = new_color;
    }
    …
```

Class example: Car (2)

```
    public void setRegistrationNumber( String registration_number ) {
        this.registrationNumber = registration_number;
    }

    public int getFuelConsumption(){
        return fuelConsumption;
    }

    @Override
    public String toString(){
        return "A " + color + " " + brand + " with number " + registrationNumber;
    }
}
```

Example: creating objects/instances

```
public class Lecture1Car {

    public static void main( String[] args ) {
        Car dacia = new Car( "Dacia", 14, "brown" );
        dacia.setRegistrationNumber( "55-XVJ-5" );
        Car bmw = new Car( "BMW", 18, "blue" );
        bmw.setRegistrationNumber( "99-JXR-4" );

        System.out.println( dacia );
        System.out.println( bmw );
    }

}
```

Running the example

In NetBeans

```
run:
A brown Dacia with number 55-XVJ-5
A blue BMW with number 99-JXR-4
BUILD SUCCESSFUL (total time: 0 seconds)
```

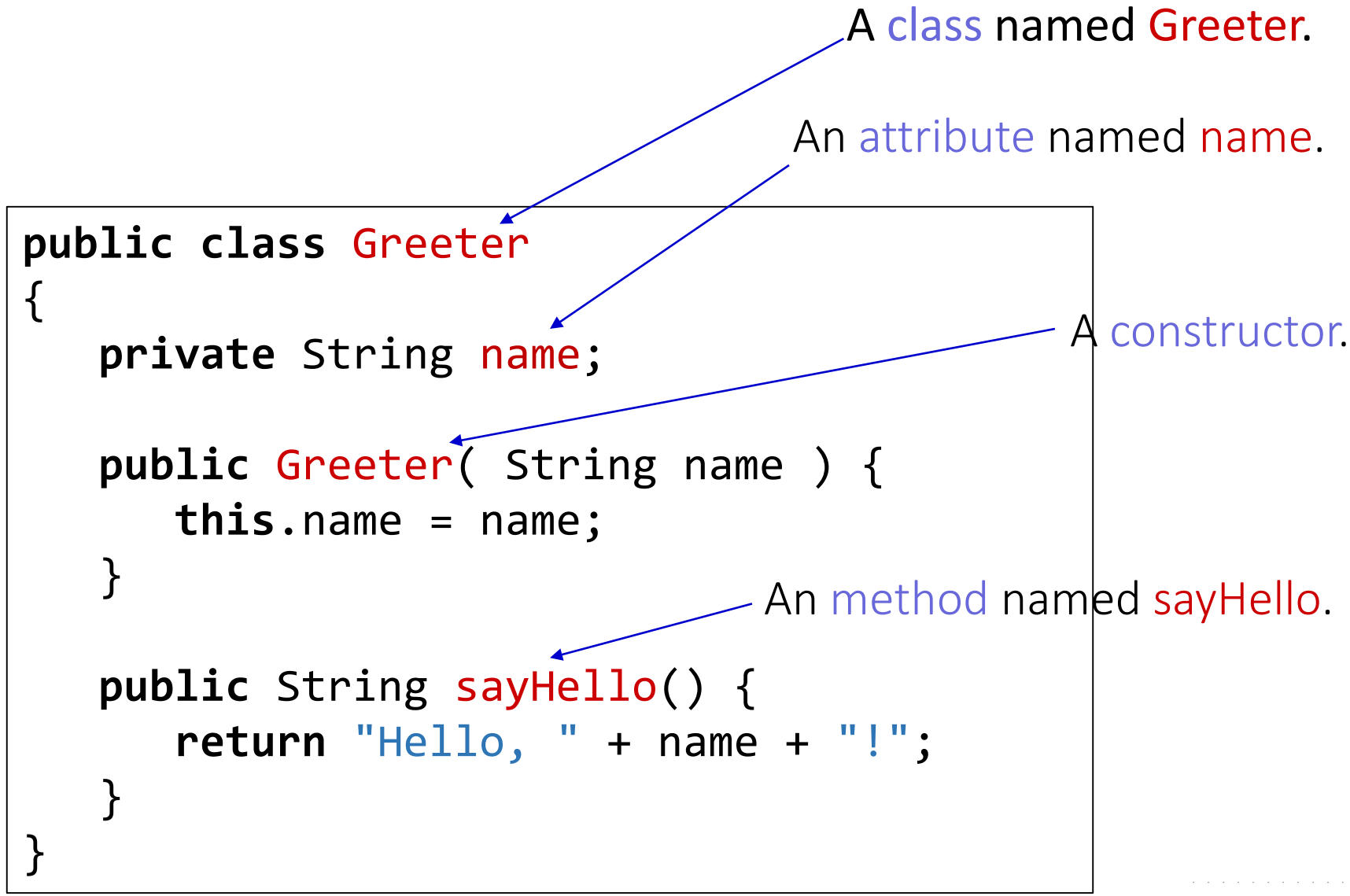
Java Class Libraries

- Java programs consist of classes
  - Include *methods* that perform tasks
- Java provides class libraries
  - Known as Java APIs (Application Programming Interfaces)
- To use Java effectively, you must know
  - Java programming language
  - Extensive class libraries
- Using Java API classes instead of writing your own versions can
  - improve program performance, because they are carefully written to perform efficiently;
  - improve program portability, because they are included in every Java implementation.

Classes

- Methods may change the values of the attributes
- Constructors (special methods) invoked implicitly (via **`new`**) to create an instance (object) of a class
- Attributes (instance variables) typically initialized by the constructor.

Example


A class named Greeter.
An attribute named name.
A constructor.
An method named sayHello.
public class Greeter
{
private String name;
public Greeter(String name) {
this.name = name;
}
public String sayHello() {
return "Hello, " + name + "!";
}
}

The Constructor

- When Java creates a new object, it calls the class's constructor.

The constructor has the same name as the class.

```
public class Greeter
{
    private String name;

    public Greeter( String name ) {
        this.name = name;
    }
```

Anatomy of a method (1)

Who can use this method?

public = anyone

protected = me and my children (subclasses)

private = only my class


```
public String sayHello() {
    return “Hello, ” + name + “!”;
}
```

Anatomy of a method (2)

What answer (value) is returned?

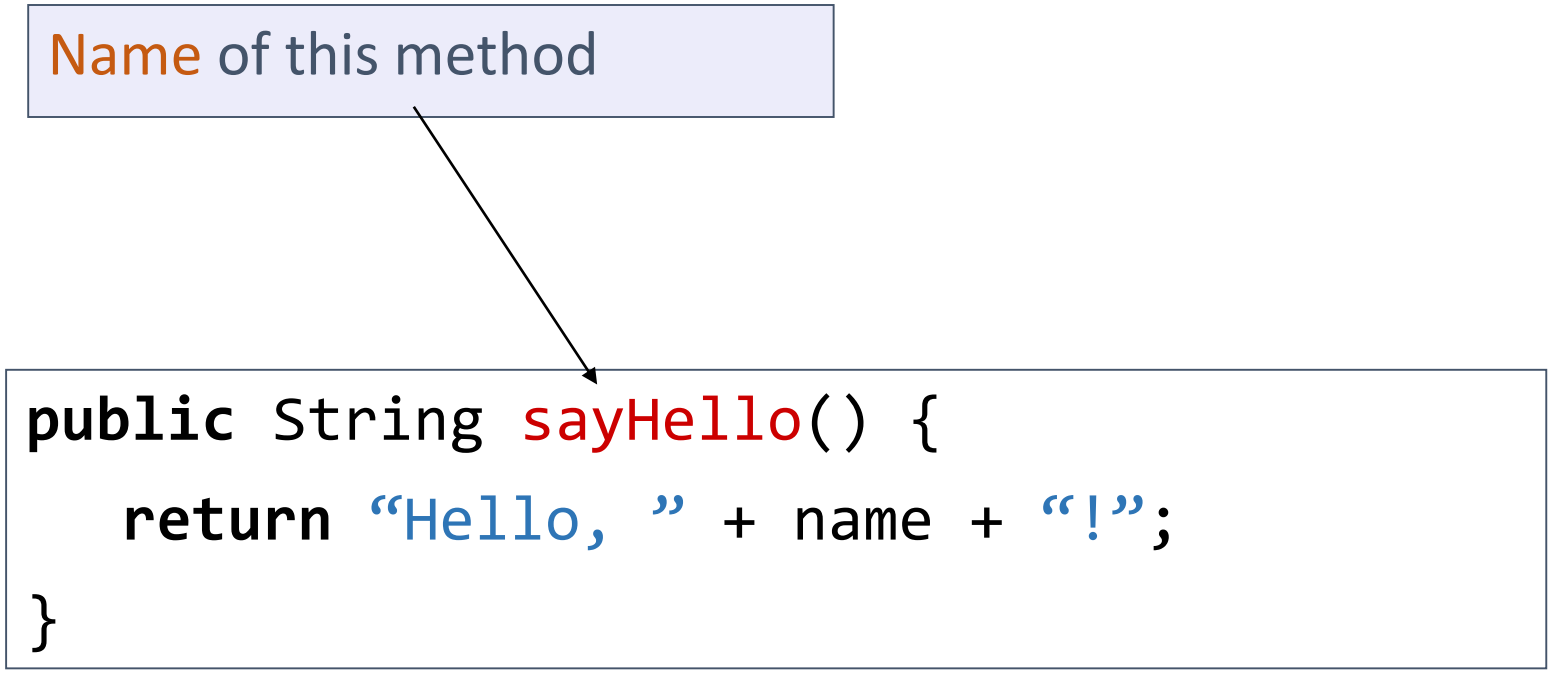
void = nothing returned

int = returns an integer (0, 1, 2, ...)

etc. a method can return *anything*

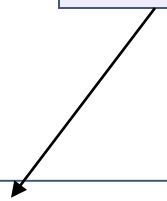
```
public String sayHello() {
    return “Hello, ” + name + “!”;
}
```

Anatomy of a method (3)



Anatomy of a method (4)

Parameters for passing info to this method (none here).



```
public String sayHello() {
    return “Hello, ” + name + “!”;
}
```

Running our example

main is the starting point for a program.

```
public class RunGreeter {
  public static void main( String[] args ){
    Greeter worldGreeter = new Greeter( "Sjaak" );
    String greeting = worldGreeter.sayHello();
    System.out.println( greeting );
  }
}
```

new creates an instance/object

Types (1)

- Two sorts of types:
  - Primitive (int, boolean, …)
  - Reference (Car, Object, Greeter,…)
- Primitive types:
  - int (4), long (8), short (2), byte (1)
  - double (8), float (4)
  - char, boolean

Types (2)

- Operations -, +, *, /, % are standard
- De `Math` class contains various handy functions (methods without side-effects) for integers and floating point numbers.

Types (3)

- Reference types:
  - classes: every class is a (reference) type.
- Special type: `String`
  - Belongs to "`java.lang.*`" API
  - Not primitive!
  - Concatenation: +

Another Example: assignment used in the past

Make a class that can generate successive prime numbers (small to large). This class should contain a method `next` yielding the next prime number in the sequence. That is, the first time `next` is called it will return 2, the second time 3, then 5, etc.

Class PrimeGenerator - Responsibilities

```
public class PrimeGenerator
{

  public PrimeGenerator() {
  }

  public int next() {
  }

}
```

- The `next` method: the first time it is called it will return 2, the second time 3, etc.
- A generator has to remember the next prime number that will be returned or the last prime that was returned

Class PrimeGenerator - attributes

```
public class PrimeGenerator
{
  private int nextPrime;

  public PrimeGenerator() {
    nextPrime = 2;
  }

  public int next() {
      …
      return nextPrime;
  }

}
```

Running/Testing the PrimeGenerator

```
public class Main {
   public static void main( String[] args ) {
      new PrimeIO().generatePrimes();
   }
}
```

`static`: no object needed to call this method point

`args`: provided by the OS

`System.in`: your keyboard

```
public class PrimeIO {
    private Scanner myScanner = new Scanner(System.in);

    public void generatePrimes() {
        System.out.print("How many prime numbers? ");
        int n = myScanner.nextInt();

        PrimeGenerator pg = new PrimeGenerator();
        for (int i = 0; i < n; i++) {
            System.out.print(pg.next() + " ");
        }
    }
}
```

`Scanner`: breaks its input into tokens

Class PrimeGenerator – helper methods

```
public class PrimeGenerator
{
  private int nextPrime;

  private boolean isPrime() {
    for ( int i = 2; i < nextPrime; i++ ) {
      if ( nextPrime % i == 0 ) {
        return false;
      }
    }
    return true;
  }

  private void findNextPrime() {
       do {
          nextPrime++;
       } while ( ! isPrime () );
  }
```

isPrime : more efficient version

```
public class PrimeGenerator
{
  private int nextPrime;

  private boolean isPrime() {
    if ( nextPrime % 2 == 0 ) {
      return false;
    } else {
      for ( int i = 3; i * i <= nextPrime; i += 2 ) {
        if ( nextPrime % i == 0 ) {
          return false;
        }
      }
      return true;
    }
  }
```

Class PrimeGenerator – complete

```
public class PrimeGenerator {
  < ... >

  public int next() {
    int thisPrime = nextPrime;
    findNextPrime();
    return thisPrime;
  }

}
```

Arrays

- An array is a special kind of object
- Think of as collection of variables of the same type
- Creating an array with 7 elements (variables) of type double

```
double[] temperature = new double[7];
```

- Initializing

  Possible to initialize at declaration time

```
double[] reading = new double[] { 3.3, 15.8, 9.7 };
```

  Mostly written as

```
double[] reading = { 3.3, 15.8, 9.7 };
```

The attribute `length`

- As an object an array has one public attribute
  - name `length`
  - Contains number of elements in the array
  - It is final, value cannot be changed

Enhanced **`for`** loop

- Enhanced **`for`** loop iterates through elements in a collection (in this case array) and provides access to each element.
  - Each iteration of the loop corresponds to an element in the array.

```
public static void main( String args[] ) {
    int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
    int total = 0;

    // add each element's value to total
    for ( int number : array ) {
       total += number;
    }

    System.out.printf( "Total of array elements: %d\n", total);
 }
```

- You cannot change the value of an array element

Multidimensional Arrays

- Arrays with arrays as elements
- Given
```
int[][] table = new int[10][6];
```
- Array `table` is actually 1 dimensional with elements of type `int[]`
  - It is an array of arrays
- Important when sequencing through multidimensional array
  - `int[][][] threeDArray;`

2D Enhanced **for** loop

```
int[][] table = { {87, 68, 94}, {100, 83, 78},
                  {85, 91, 76}, {87,  23, 44} };
int total = 0;

for ( int[] row : table ) {
    for ( int number : row ) {
        total += number;
    }
}
```

Ragged Arrays

- Not necessary for all rows to be of the same length
- Example:

```
int[][] b;
b = new int[3][];
b[0] = new int[5]; // First row, 5 elements
b[1] = new int[7]; // Second row, 7 elements
b[2] = new int[4]; // Third row, 4 elements
```

Example: Binomial Coefficients

binomial coefficient:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad \binom{7}{3} = \frac{7!}{3!(4)!} = \frac{5040}{6 \times 24} = 35$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \qquad \binom{n}{0} = \binom{n}{n} = 1$$

Pascal's Identity

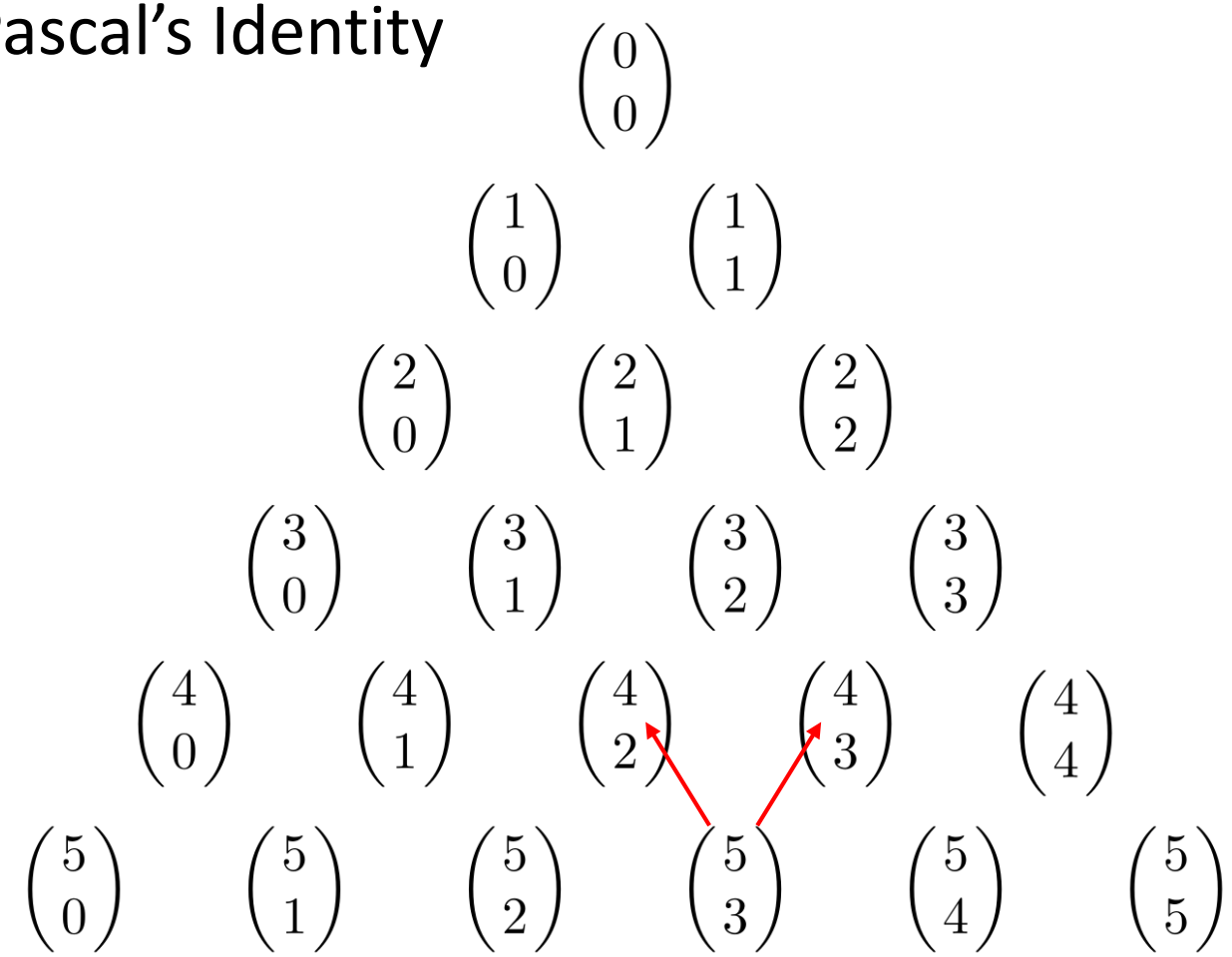
Binomial coefficients in JAVA

```
private static int choose( int n, int k ) {
   if ( k == 0 || k == n ) {
      return 1;
   } else {
      return choose( n-1, k-1 ) + choose( n-1, k );
   }
}
```

Double recursion:
(very) inefficient!

Pascal's Triangle

- Geometric arrangement of binomial coefficients
- Based on Pascal's Identity



$$\binom{n}{0} = \binom{n}{n} = 1$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Efficient solution using arrays

Dynamic programming

```
public class PascalsTriangle {
    private int[][] myBinomial;

    public PascalsTriangle( int N ) {
        myBinomial = new int[N+1][];
        for ( int n = 0; n <= N; n++ ) {
            myBinomial[n] = new int[n+1];
            myBinomial[n][0] = myBinomial[n][n] = 1;
            for ( int k = 1; k < n; k++ ) {
                myBinomial[n][k] = myBinomial[n-1][k-1] + myBinomial[n-1][k];
            }
        }
    }
    public int choose( int n, int k ) { return myBinomial[n][k]; }
}
```

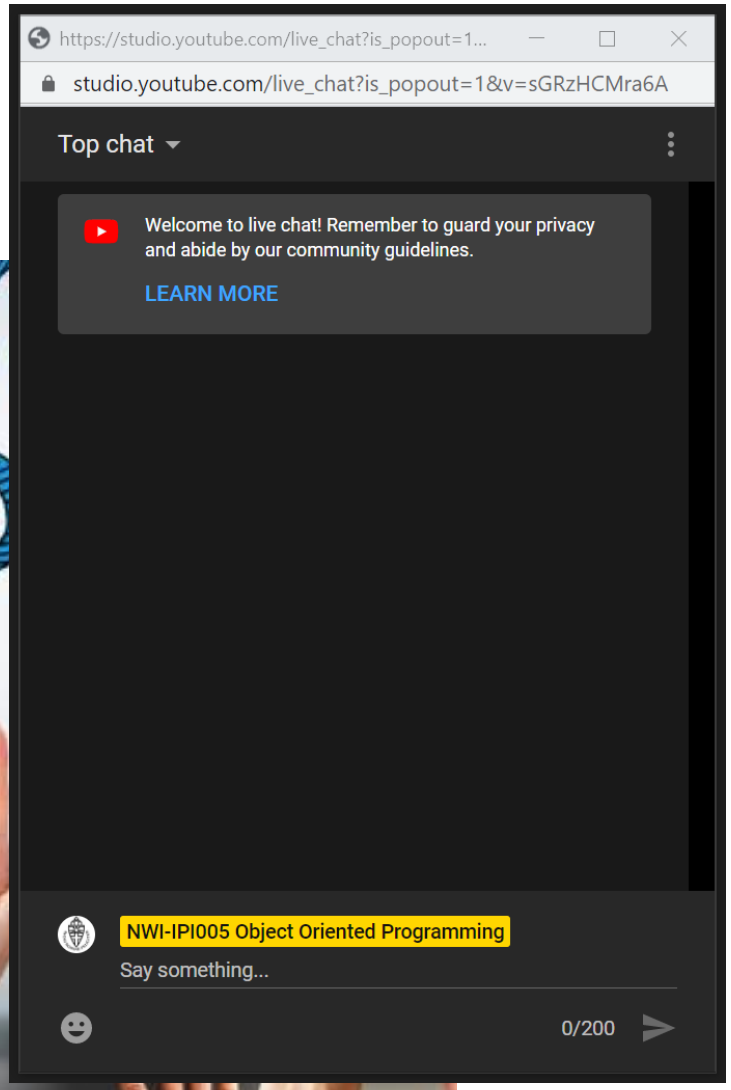
Running the example

```
public class Binomial {

    public static void main( String[] args ) {
        PascalsTriangle pt = new PascalsTriangle( 20 );
        System.out.println( pt.choose( 7, 3 ) );
    }

}
```

Next lecture

Java + Interfaces

Finally


https://studio.youtube.com/live_chat?is_popout=1...
studio.youtube.com/live_chat?is_popout=1&v=sGRzHCMra6A
Top chat
Welcome to live chat! Remember to guard your privacy and abide by our community guidelines.
LEARN MORE
NWI-IPI005 Object Oriented Programming
Say something...
0/200