tutorial OO 1
Starting with Java

Pol Van Aubel

Radboud University

PRELIMINARIES

BETER GOED GEJAT
DAN SLECHT VERZONNEN

DACHT IK, VLAK VOORDAT IK
VAN DE UNIVERSITEIT WERD GETRAPT

– Unknown, ca. 2010

plagiarism

- **Don't *ever* submit another's work as though it was your own!**
- Not in this course, not in other courses, not after your studies
- Plagiarism is a serious academic offence
- We **are obliged** to report you to the exam committee and the dean
- It **will result** in exclusion from the exam
- And even so, every year we detect a dozen cases

plagiarism

- Working in teams is fine
- Collaboration and discussion within your team is fine
- Discussion with other teams is fine
- Getting help when stuck is fine
- Getting inspiration from others is fine
- Reading documentation is fine
- Consulting books or online resources is fine
  - (Posting literal homework assignments on Stack Overflow is **NOT fine**)

- But **ALWAYS WRITE YOUR OWN CODE**

asking questions / troubleshooting

- Questions about the assignments and/or Brightspace
  - Student Assistants during the lab
  - Liye Guo <l.guo@cs.ru.nl>

- Questions about the course, exam, scheduling, administrative stuff:
  - Sjaak Smetsers <s.smetsers@cs.ru.nl>
- Questions about the tutorials I teach:
  - Pol Van Aubel <pol.vanaubel@cs.ru.nl>
  - **Don't** expect us to answer within minutes or outside of working hours
  - You have plenty of time to ask questions during lab sessions, after or during lectures, or by e-mail on Monday – Friday

asking questions / troubleshooting

- A question should provide:
  - a description of your actions (what did you do?)
  - a description of the specific symptoms (what was the **exact** result?)
  - a description of why those symptoms are adverse
    - (what were you expecting to happen instead?)

- Of course we will help you if you can't provide this
- But it saves everyone's time if you manage to provide this up front
- And might even lead you to the answer before ever asking us

- Example of a time-consuming question: "Help, it doesn't work."

course enrolment

- You should be enrolled by now
- If not, go to the student desk and enrol
- If you run into trouble doing this (need permission), e-mail Sjaak and Liye

structure of the tutorial (werkcollege)

1) Important announcements, administrative stuff
2) Last week's assignment, common problems, tricky stuff, etc.
3) Current assignment, explanation of techniques not covered in the lecture
    - (in principle more about the practical side of things than the lecture)
4) If applicable: demonstration of IDE functionality

- Might take just an hour, might take the full tutorial time
- I will teach only the first few weeks of tutorials
- This structure is how I'm going to do it, Sjaak might do it slightly differently

- I **expect** you to **be prepared:**
  - having read the assignment, ready to ask questions; also ask questions you want me to cover on Discord the day before

ASSIGNMENTS

goal of the assignments

- practical experience, only understanding the lecture material is insufficient
- if you are prepared the assignment can be done in 4 hours
  - study lecture slides (read book by need)
  - study assignment (ask clarification in tutorial)
  - make a rough design before you start
- always check and verify you understand the teaching goals!
  - on the exam you have to **apply** ***techniques*** in a ***new*** situation
- assignments are a teaching situation: **insufficient (onvoldoende) is an option**
  - **insufficient means there is work to be done to pass the exam!**
  - show you did a serious attempt: otherwise FAIL: two retries during the course
    - (program must compile, all essential parts must be present)
- the course material is not completely covered by the assignments:
  do not forget to study the lecture slides!

OO = tools + how to use them

- **understand** concepts in OO programming
- **apply** those concepts to structure programs

topic	week 1	week 2	week 3	week 4	week 5	week 6	…
classes	✓	✓	✓	✓	✓	✓	✓ …
encapsulation	✓	✓	✓	✓	✓	✓	✓
interfaces	✗	✓	?	?	?	?	…
inheritance	✗	✗	✓	?	?	?	…
collections	✗	✗	✗	✓	?	?	…

master the tool, clearly stated

apply the tool if appropriate

vague?

range of assignments

dull once you master the technique

fun, but often more than one solution

design and make a comfortable desk chair

detailed instructions to master technique

global requirements to master design

work in pairs

- a partner can help if you are stuck
- discussion with a partner helps understanding
- together you learn more
  - this does not work if you alternate making the assignments …
- choose a partner of equal strength
  - otherwise one of you will only sit there and watch the other learn how to program
  - in the end YOU have to program at the exam
  - you're allowed to switch partners
- working in pairs reduces the graduation work

group enrolment

- Brightspace can handle only 200 groups in a category
- Therefore there are two categories
  - Assignments A
  - Assignments B
- It does not matter which category you enrol in!
- Find an empty group and enrol yourselves
- In case of TOCTOU conflict, un-enrol and pick a different group
- Groups will be frozen on the day of the assignment deadline
- Contact Liye Guo, <l.guo@cs.ru.nl>, to change group
- Detailed information in Assignment 1, section 5.1

handing in assignments

- Deadlines are listed on Brightspace, usually Sunday, 23:59, same week
- Handing in only possible when enrolled in a group
- Even if working alone!
- Detailed submission instructions in Assignment 1, section 5.2
- Netbeans: Export Project → to ZIP
- Other IDEs: use export function if available, or zip the project folder
  - (This might change if we notice problems)
- Do not submit RAR, GZIP, 7zip, TAR, or anything other than normal ZIP
- If you do, you **FAIL** the assignment, wasting a retake!
- Windows: Right-click, “send to” → “compressed folder” (or similar)
- Mac OS: Right-click, “compress”
- Linux: “zip -r assignment.zip folder”

“Deadline on Sunday”

!=

“Support in the weekend”

grade reuse

- If you took the course before, you **are** allowed to reuse **sufficient** grades
  - (or higher grades, of course)
- Conditions:
  - Assignment is the same as the one you are reusing. Check this!
  - We may state that an assignment is considered different, even if similar.
  - Submit the correct assignment! Order may have changed.
- To reuse, submit a **plain text** file (.txt) containing:
  - Name and student number
  - Number and year of the assignment you want to reuse
  - The grade you got

why you might not want to reuse a grade

- We don't provide feedback for our own amusement
- Feedback is supposed to help you pass the exam
- Build and submit a new solution if
  - You want feedback
  - You feel like practising to master the learning goals
  - You're working together with a teammate who hasn't done the assignment
    - (They may value and learn from your experience and insight, but please don't make the assignment for them)

testing your work

- We test your work in two ways:
  - Test framework is provided with the assignment (I will demo it later)
  - PersonalProf analyses your code for common shortcomings on submission
- The test framework tests whether your code's behaviour is correct
- Needs glue code to go between your code and our tests that call the glue
- Peeking at the code of the tests might "inspire" your design
- My advice: Don't look at the tests, only the glue, unless you really need to
  - (e.g. the glue code isn't working and you don't understand why)
  - (or you keep failing a test you don't understand)
- These tests are for your benefit, try to make sure you pass them

PersonalProf

- Code analysis upon submission
- Report is immediately available to you as .txt
  - E.g. "Missing class Group in file:///" means you haven't implemented a required class!
- Submit on time, so you can fix these issues!
- If your solution is different but you're confident it's good, ignore PersonalProf

See https://brightspace.ru.nl/d2l/le/content/163566/viewContent/1094029/View for more details.

Passing the test cases and PersonalProf does not guarantee a **sufficient** grade

But it does mean you won't fail on trivial matters

IDE

Integrated Development Environment: IDE

- very convenient to develop Java programs
  - knows the methods of classes, also from the library
  - number and type of arguments
  - syntax highlighting
  - error indication
  - automatic addition of imports
  - generation of getters and setters
  - generation of test skeleton
  - interactive debugger
  - project management
  - ...

Java IDEs

- we propose Netbeans
  - https://netbeans.org/
  - free installation of 12(.2) on your own machine
  - fairly simple
  - sufficiently powerful
  - installed at Science machines, used to test example code
  - **available at the exam**

- alternatives:
  - **IntelliJ**: free community edition
    https://www.jetbrains.com/idea/

eclipse:
https://www.eclipse.org/eclipseide/

Java version

- The book is based on Java 8
- Most of the lecture material works in Java 8
  - But we *might* use more recent functionality!
- Assignments are built using Java 15 (or 13)
  - (In particular, OpenJDK, so no need to install Oracle JDK on Linux)
  - Errors if you try opening the provided Assignment projects with older Java
- Java 15 is backwards compatible with Java 8
- So, install JDK 15

your Java project

- a group of source files and the settings to build, run, and debug a program
  - in Java each **class** should **have its own file**
  - the IDE provides convenient tools to change settings
- all Java development has to take place within a project
- large application can be split in several projects
  - for this course one project per assignment will do
- the IDE has several project templates
  - for the time being we need only `Java Application`
  - the IDE creates a set of directories and files for each new project

ASSIGNMENT 1

techniques of this week

- class definition
  - attributes (private)
  - methods (private/public)
  - constructor
  - `toString`
  - `equals`
- compile and execute Java program
- Java arrays
  - definition, length
  - index, update
- basic I/O
  - `System.out.println`
  - `Scanner` + `System.in`

a fresh Java Application project contains 1 class

```
/*
 * To change this header, choose License Headers in Project Properties
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package test;

/**
 *
 * @author pieterkoopman
 */
public class Test {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

program execution of project starts here

hello world

```
public class Test {

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {
    System.out.println("Hello Nijmegen");
  }
}
```

even basic output belongs to a class

adding a class

- the IDE also has templates for this
- we need a Java Class

```
package test;

/**
 *
 * @author pieterkoopman
 */
public class MyClass {

}
```

in file
MyClass.java

Object Oriented design

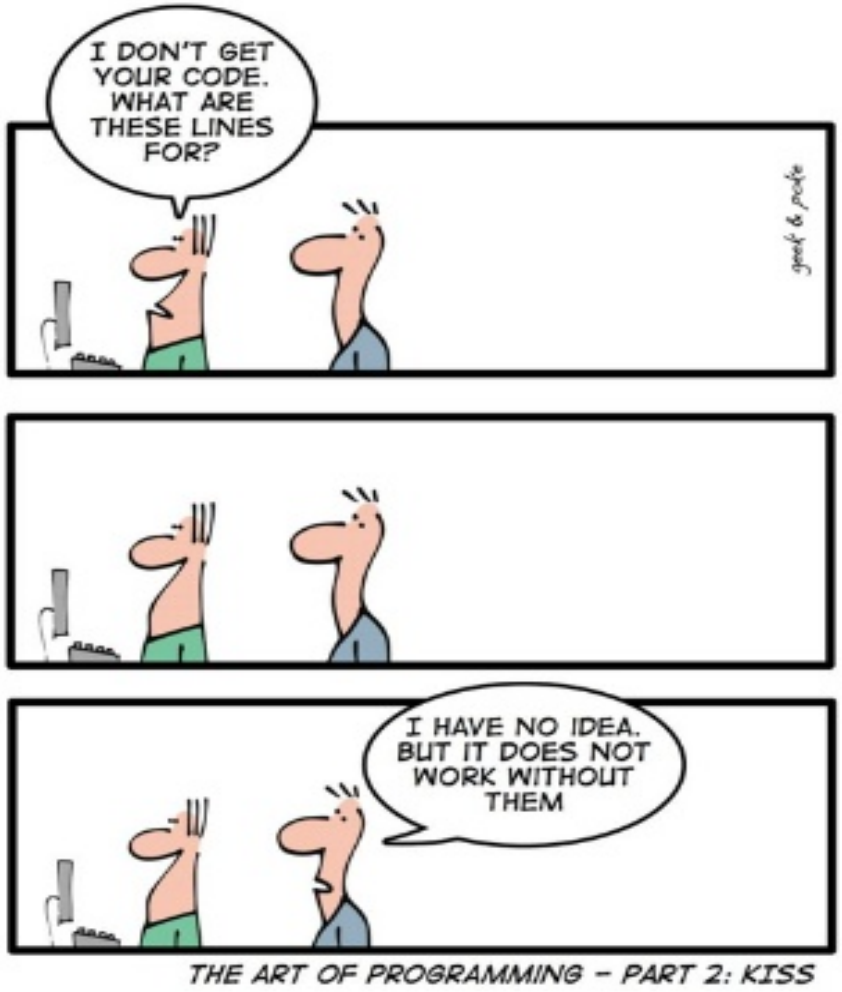
Object Oriented Principles

- classes are the main building block
  - objects are instances of these classes
  - a class is the **building plan** of objects
- encapsulation
  - objects should have as little interaction as possible
  - bundle data inside an object with methods to manipulate this data
  - all attributes are private, only getters and setters if useful
- separation of concerns
  - use different classes for unrelated storage and manipulations
- localized I/O
  - plan any class to be suited for reuse:
    no I/O by ordinary classes, all I/O localized
  - separate I/O makes automatic testing much easier

other important guidelines

- KISS
  - Keep it Short and Simple
  - Keep it Simple, Stupid
  - Einstein: "Make everything as simple as possible, but not simpler"
  - 'clever' is a bad for maintenance

- DRY
  - Don't Repeat Yourself:
    no copies of code fragments;
    introduce a new method instead
  - repetition is bad for maintenance



“Why use a do-while loop?”

```
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 () );
  }
```

DRY!

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

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

which classes to define?

- identify main kinds of 'things' in your application
  - student, group, point in R^2
  - each kind of things becomes a **class**
- identify instances of these things: objects
  - the students Alice and Bob
- what are properties of these classes: **attributes**
  - what do they know
  - each student has a name of type String and a number of type int
- what can the objects do, which properties can be obtained, which properties can be changed: **methods**
  - add a student to a group
  - convert attribute values to string
  - compute distance to other point

encapsulation

- hide data within objects
- make all attributes private (only known inside the class)
  - separation of concerns + control over changes
  - the only exception can be constants

```
public class Point {
  private int x, y;

}
```

getters and setters

- obtain and change a selection of attributes
  - the IDE can generate those getters and setters

```
public class Point {
  private int x, y;

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  public void setX(int x) {
    this.x = Math.max(x, 0);
  }

  public void setY(int y) {
    if (y >= 0) {
      this.y = y;
    }
  }
```

methods control the values of attributes

each method belongs to a class

constructors

- constructors assign initial values to attributes
  - nothing else needs to be done in the constructor

```
public Point(int x, int y) {
  this.x = x;
  this.y = y;
}

public Point() {
  x = 0;
  y = 0;
}
```

`this.x` the attribute of this object

as many constructors as you need

rule of thumb: define at least one constructor

toString()

- in Java you never make a method that prints an object, instead convert the object to a string and print this
  - redefine the predefined toString method of the class

```
@Override
public String toString() {
  return "(" + x + ", " + y + ")";
}
```

redefine method

implicit conversion to string

for objects the `toString()`

tailor made methods

- define methods for all class specific operations

```
public double distance (Point q) {
  double dx = q.x - x;
  double dy = q.y - y;
  return Math.sqrt(dx * dx + dy * dy);
}
```

inside the class it is not necessary to use getters

a main class

```
public class PointProject {

  private Point p;

  public PointProject(Point p) {
    this.p = p;
  }

  private void run() {
    Scanner scan = new Scanner(System.in);
    Point origin = new Point();
    for (String s = scan.nextLine(); ! s.isEmpty();
                                     s = scan.nextLine()) {
      switch (s) {
        case "u":
        case "up":
          p.setY(p.getY() + 1);
          break;
        case ... other commands
        default:
          System.out.println("Unkown command");
      }
      System.out.println(p + " on distance " + p.distance(origin));
    }
  }
```

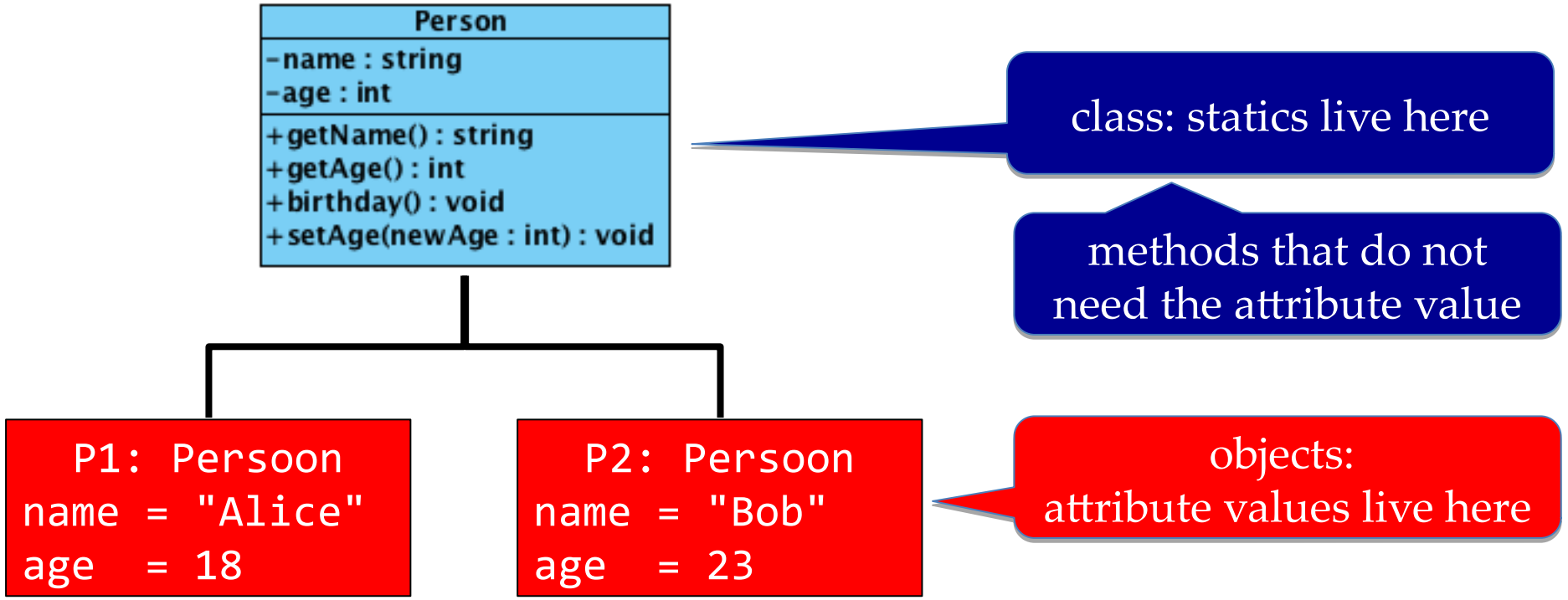
static vs dynamic

- static methods and attributes belong to the class
  - use the keyword `static` in the definition
  - e.g. methods that do not need attribute values
  - an attribute that counts the number of instances created,
    static attributes should be very rare!
- all other (dynamic) attributes and methods belong to object: instances of this class
  - make objects with `new`
    ```
    Point origin = new Point();
    ```
  - how to obtain the first object?

    in the `static main` method!

static / dynamic

- static belongs to class
- dynamic to object: instance of class



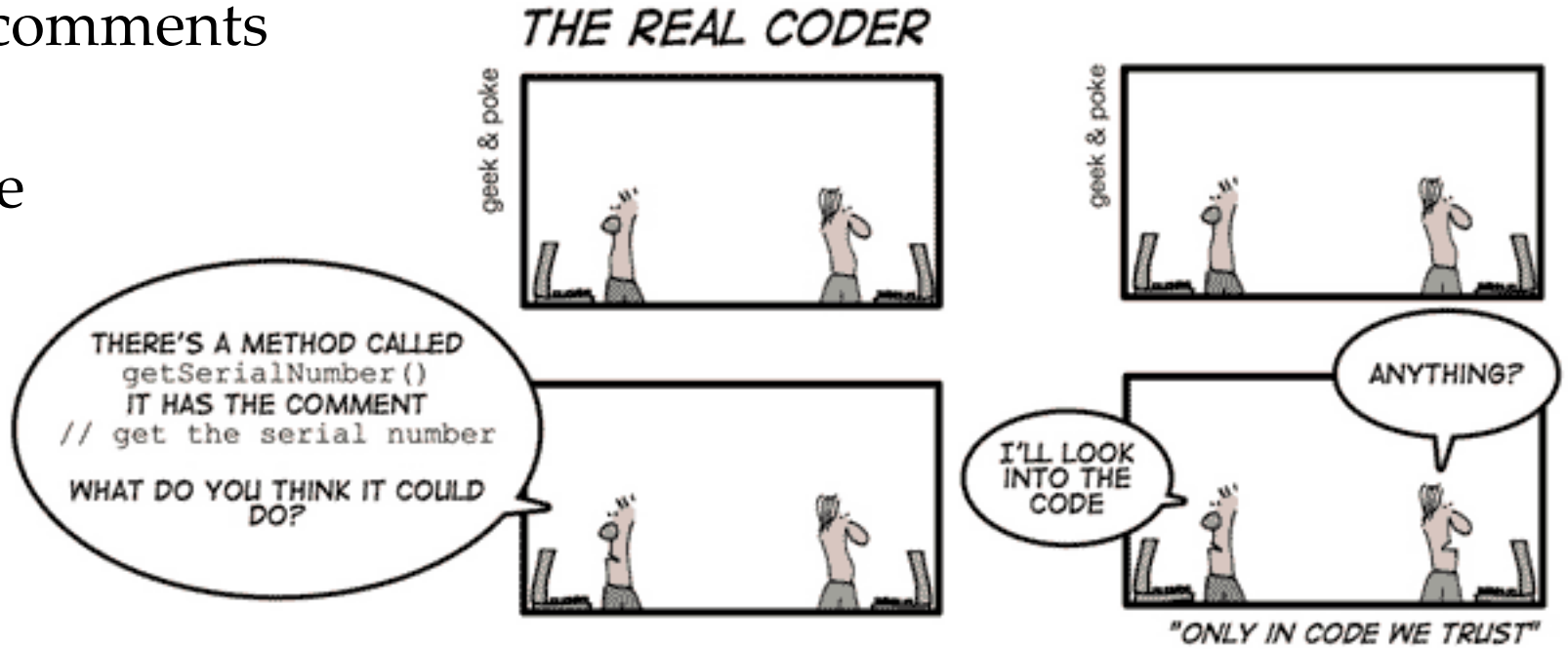
a main for our point example

```
public static void main(String[] args) {
    Point p = new Point(1, 1);
    PointProject pp = new PointProject(p);
    pp.run();
  }
```

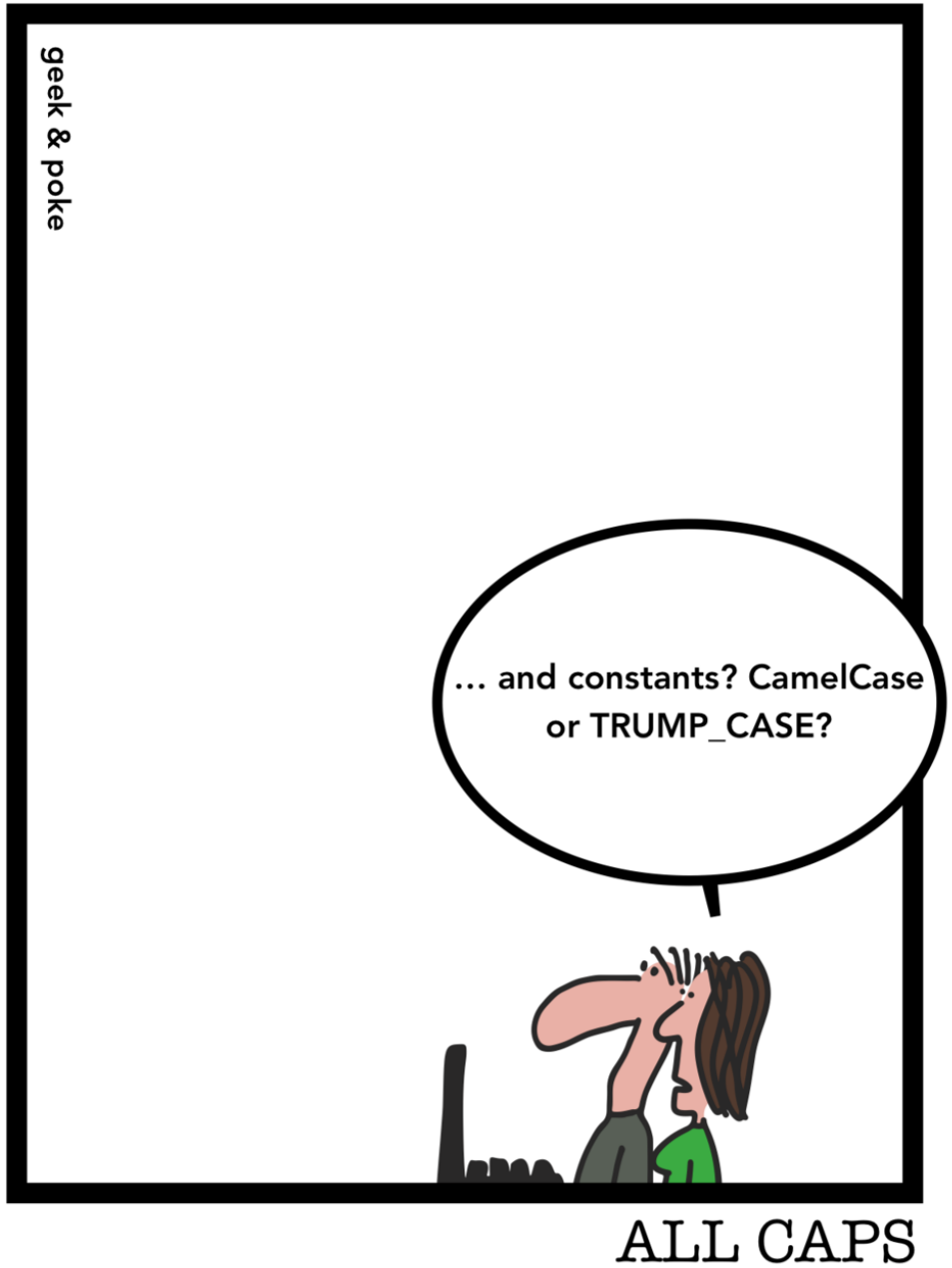
- it will work fine to put all code of run in constructor
- this is considered to be a **BAD** style!
- the constructor should only initialize the attributes

javadoc

- standard way to write comments
- a tool converts these comments to a webpage documenting the class
- enables reuse



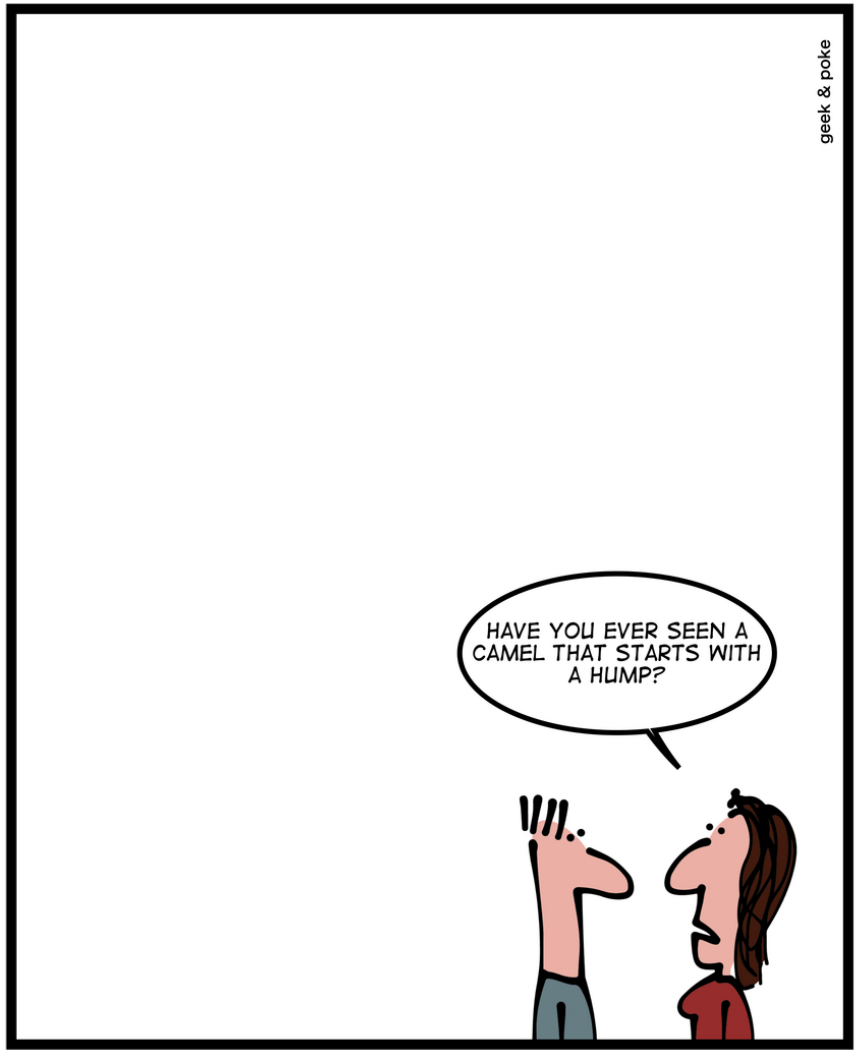
- in this course:
  - `@author`
  - short description of very smart methods



naming conventions

- classes: CamelCase start with upper case
- methods: camelCase start with lower case
- attributes: idem
- constants: TRUMP_CASE

SORRY C# FOLKS!



PROBLEM SOLVED:
METHOD NAMES START LOWER CASE,
PERIOD!

layout

- make your code nice and easy to read
- use the guidelines provided on Brightspace
- BAD

```
int ugplypower(int a ,int b )
{int x=a,n=b ,r=1;
   while (n!=0 )
  if(n%2==0) {x=x*x;n= n/2 ; }

  else { r=r*x ;n=n-1; }
      return r;}
```

use Netbeans
Source > Format
to improve layout

better programming style

```
/**
 * computes {@code base} to the power {@code exp}
 * @param base value of base
 * @param exp  value of exponent
 * @return {@code base^exp} if {@code exp >= 0}
 */
public static int power(int base, int exp) {
  int result = 1;
  int b = base;
  // loop invariant: 0 <= n <= exp && result * b^n == base^exp
  for (int n = exp; n > 0; ) { // n >= 0
    if (n % 2 == 0) {          // even exponent
      b = b * b;
      n = n / 2;
    } else {                   // odd exponent
      result = result * b;
      n = n - 1;
    }
  }
  return result;  // n == 0, hence result == base^exp using the invariant
}
```

KISS

equality in Java 1

```
private void test2 () {
  int x = 42;
  int y = 42;
  System.out.println("x == y yields" + (x == y));
}
```

```
x == y yields true
```

- equality on basic values works as expected

equality in Java 2

```
private void test3 () {
  String x = new String("hello");
  String y = new String("hello");
  System.out.println("x == y yields " + (x == y));
}
```

```
x == y yields false
```

- this equality does not work as you might expect
- different String objects, == checks object identity
- use method `equals` to compare object values

equality in Java 3

```
private void test4 () {
  String x = new String("hello");
  String y = new String("hello");
  System.out.println("x.equals(y) yields " + x.equals(y));
}
```

`x.equals(y) yields true`

- different String objects, `equals` checks content

equality in Java 3

- Java has many strange corners, e.g.

```
private void test3a () {
  String x = "hello";
  String y = "hello";
  System.out.println("x == y yields ” + (x == y));
}
```

```
x == y yields true
```

- compiler optimisation !

immutable objects

STRINGS

String & StringBuilder

- **String** is a class in Java, not a basic type
- you can inspect the contents of strings

```
s.length ( )
s.charAt ( 3 )
s.indexOf( 'e', 3 )
```

- Strings are **immutable** objects !
- use `StringBuilder` to change String-like objects
- there is a very similar type `StringBuffer` in Java
  - we will discuss details later

strings are immutable

`s.toLowerCase ( )`

- returns a new string object

```
String s = "a string";
s.toUpperCase();
System.out.println(s);
```

- yields: `a string` not `A STRING`

`"appel".concat("flap")`

- returns a new string object `"appelflap"`

String & StringBuilder 2

- use `StringBuilder` to change strings
- has most methods of `String`
- there is a very similar class `StringBuffer`
- they have special methods like:

```
sb.append ( '.' )
sb.append ( 5 )
sb.append ( "appelflap" )
sb.setCharAt ( 3, 'e' )
```

reading from system.in

SCANNER

terminal input

- in C++ we have `cin`
- in Java we have `System.in`
  - this is not the most convenient input tool
- `System.in.read()` reads the next byte
- the scanner is a more convenient input tool

```
Scanner scanner = new Scanner (System.in);
```

- the class `Scanner` provides more convenient input methods

some scanner examples

```
Scanner scanner = new Scanner (System.in);
int x = scanner.nextInt();
```

- scans the next token of the input as an int

```
String name = scanner.nextLine();
```

- returns the rest of the current line
- after reading an `int`, this is often an empty string ☹
- better: read a nonempty sequence of non-spaces
  - using a regular expression (see http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html)

```
String name = scanner.next("\\S+");
```

pattern is a regular expression

`scanner.next()` will work fine here

OO DESIGN RECAP

classes; attributes + methods

classes the kind of 'things'	`Student`	`Group`	`Main`
attributes properties of class	`name` `number`	`students` `size`	

methods
behaviour,
what can we do with the objects
	`toString`
	`changeName`
		`toString`
		`addStudent`
		`getSize`
		`selectStudent`
		`isFull`
			

initialized in the constructor

fine to add classes attributes and methods during development

IDE DEMO