cphbusiness

COPENHAGEN BUSINESS ACADEMY





IT Seminar 2016

Dream team:
Adam Lewandowski
Ebbe Vig Nielsen
Kasper Olesen
Teodor Mihail Costica

Anders Kalhauge
Kasper Østerbye

cphbusiness

Short about our institution and programs

Cphbusiness - what

We are what is known as a college, or undergraduate school.

Technically, we educate to the first cycle in the bologna model

Our goal is to graduate students to work in the Danish industry.

Cphbusiness – what II

- 5 locations in the North of Copenhagen
- 3000 full time students
- 600 full students at the location where I work
  - Computer Science
  - Multimedia Designer
  - Market Economists
  - Service Economists
- All programs can be taken in English, and most in Danish as well.
- Both 2½ year and 1 ½ top up (full Bachelor)

Cphbusiness – Computer Science

- The students
  - 250 full time students
  - Average age is 25
  - Only 5% is female

- Focus:
  - Computer Science as an applied science
  - Designing and building of software systems
  - Focus on large scale systems
  - Focus on distributed development
  - Focus on development methods and business understanding

Computer Science program AP

5th Semester
Internship & Thesis

4th Semester
3 Electives
(Android, Arduino, .Net, Game AI, Adv. java, Mean, Entrepreneurship, Unity)

3rd Semester
Modern web based systems
(Rest, deployment, client side programming, system development, larger project)

2nd Semester
Classic web based systems
(Databases, Web servers, Servlets, algorithms, Scrum, Test, Business understanding)

1st Semester
Single user systems
(object oriented programming, files, GUI, test, projects)

cphbusiness

Computer Science Bachelor

3rd Semester
Internship & Thesis

2nd Semester
Development of Large Systems
Systems Integration
Electives

1st Semester
Contract Based Systems Development
Databases for Systems Developers
Test

cphbusiness

Cphbusiness - where

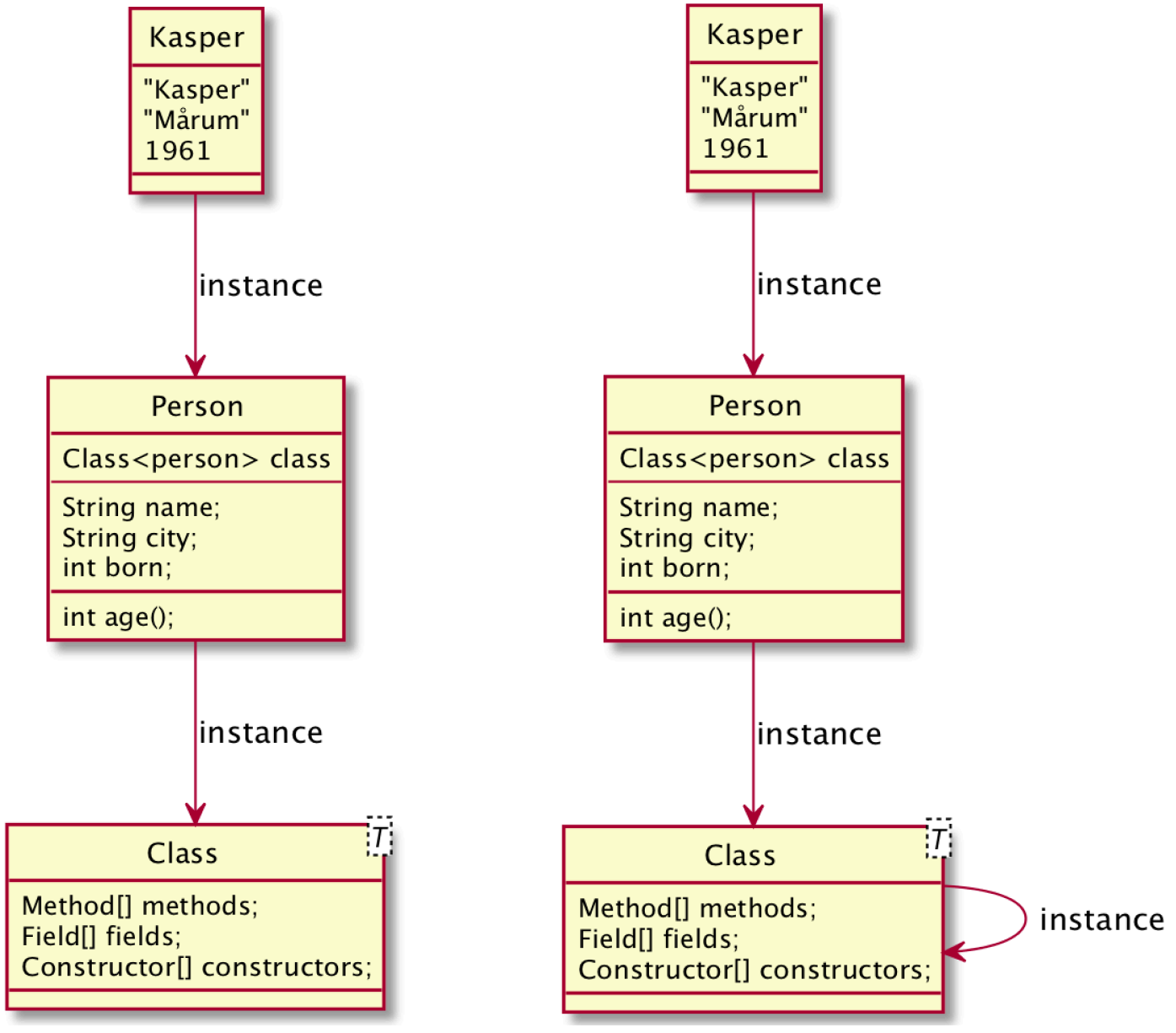
cphbusiness

cphbusiness



Java reflection

Instance of relationships



Navigating the meta level

- Get hold of the class object
  - Either as **Person.class**, or
  - as obj.getClass()
- Get hold of methods using
  - cl.getDeclaredMethods() – all methods in a class
  - cl.getMethods() – all public methods (incl. inherited)
- Get hold of the fields using
  - cl.getDeclaredFields() – all fields in a class
  - cl.getFields() – all public fields (incl. inherited)

Example – API for Method

- Class<?> getReturnType()
- Class<?>[] getParameterTypes()
- String getName()
- Object invoke(Object obj, Object... args)
- Annotation[] getDeclaredAnnotations()
- Class<?> getDeclaringClass()
- Class<?>[] getExceptionTypes()

- boolean isAccessible()
- void setAccessible(boolean flag)

Invoking a method using reflection

given a method object mm representing the method setName in class person

```
class Person {
    String name;
    String city;
    public void setName(String n){
        this.name = n;
    }
}
```

one invoke such a method as

mm.invoke(pObj, “Kasper”)

to do what without reflection is pObj.setName(“Kasper”);

-- remember that all non-static methods need a “this” reference to work.

Demo in Netbeans

A word about efficiency

Invoking a method (given a method object)

- Fast (nearly the same as regular call)

Finding a method using reflection

- Can be costly (10-15 times as expensive)

Notice this simple and efficient usage:

```
Map<YourIndex, Method> yourCache;
yourCache.get(…).invoke(...)
```

Demo in Netbeans

Exercises

a) Write a method (it does not matter where you put it, it could be a static method in a class with just a main method). The method should have the following signature:

```
Object callMethod(Object obj, String methodName)
```

The *callMethod* should return the result of calling the method named *methodName* on the object *obj.* If something goes wrong (there is no such method, the method is private,…), *callMethod* should return *null.*

b) Extend the solution in a) to allow parameters on the method. The signature could be:

```
Object callMethod(Object obj, String methodName, Object… args)
```

Exceptions should be handled as before, and *callMethod should* return *null* in case an exception is thrown.

c) Write a method which can be used to read the value of a field. It can be given the signature:

```
Object getField(Object obj, String fieldName)
```

The method *getField* should return the value of the field *fieldName* from the object *obj.* As in a) and b), catch all the possible exceptions and return null if getField fails for some reason.