Introduction to Object-Oriented Programming

Nancy Hitschfeld
Matías Toro

The world without OOP

Functional/Algorithmic Decomposition

each module is a STEP in an overall process

aka. structured programming/design

highlights process structure

Example:

> “display some shapes whose description are in a database”

Functional decomposition

Recipe

1. Get list of shapes in DB
2. Display each individual shape on the monitor
   a. Identify the type of shape
   b. Get the location of the shape
   c. Call the appropriate function that displays the shape given its location and description

2.c) Call the appropriate function that displays the shape given its location:

```
def displayShape(s: Shape): Unit {
  if(s.tag == "Square") {  /*...display for square...*/ }
  else if(s.tag == "Circle") {  /*...display for circle...*/}
  ...
}
```

what if we need to support a new kind of shape?

What if shapes have different descriptions?

Problems

```
def displayShape(s: Shape): Unit {
  if(s.tag == "Square") {  /*...display for square...*/ }
  else if(s.tag == "Circle") {  /*...display for circle...*/}
  ...
}
```

Weak cohesion

"how closely the operations in a routine are related"

here, function does too much

Tight coupling

"strength of connection between two routines"

here, changes in the description of a shape affect all functions that use it

Unwanted Consequences

Changing a function, or a piece of data used by a function, has an unexpected impact on other pieces of code.

A function that touches many different pieces of data is more fragile wrt change

```
def displayShape(s: Shape): Unit {
  if(s.tag == "Square") {
    ... s.position.x ...
    ... s.position.y ...
  } else if(s.tag == "Circle") {  /*...display for circle...*/}
  ...
}
```

accesses the representation of
different structures

- new kind of shape
- new representation of shape (description)
- new representation of position (eg. polar)

Functional decomposition

Decomposing in functional steps

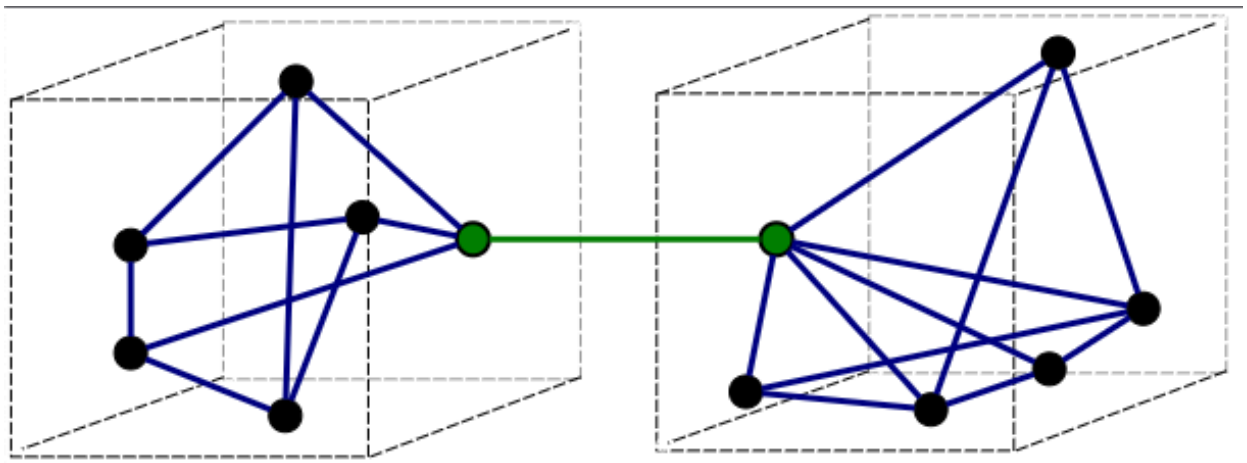
- natural (recipe)
- tends to lead to one big “main” program responsible for controlling the whole flow
- “with great power comes great responsibility”
- does not help much with dealing with change
  - new shapes, new ways to display shapes
  - centralized logic is likely to break with change

We want:

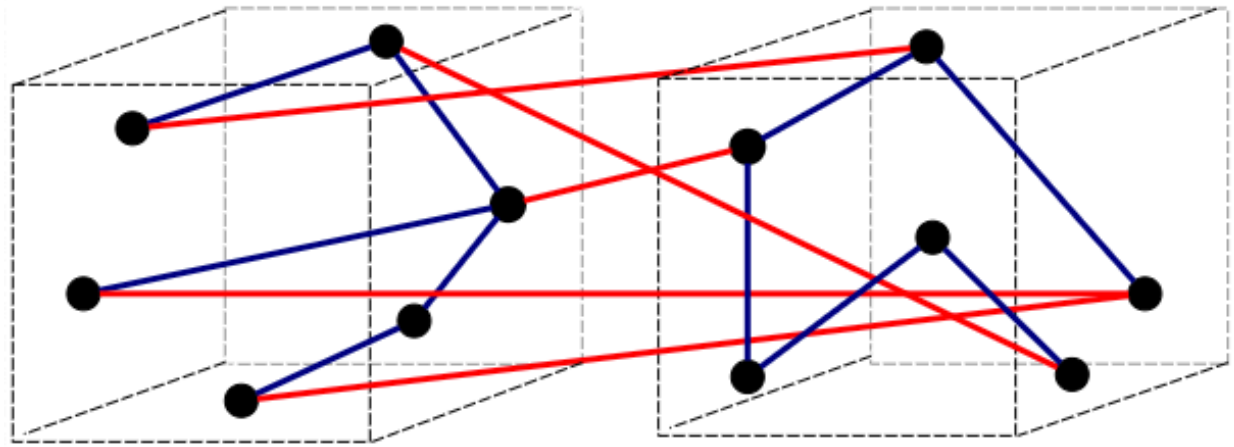
High cohesion
(internal integrity)

Low coupling
(small, direct, visible, flexible)

Coupling and cohesion



a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

OOP: Shifting Responsibilities

Shifting Responsibilities

1. Ask DB for the list of shapes
2. Ask each shape to display itself

Same things must be implemented, but very different organization

client no longer responsible for everything

shapes are responsible for their own behavior

Comparison

A- Explicit instructions for display

client has to pay attention to everything

client is responsible for everything

```
def displayShape(s: Shape): Unit {
  if(s.tag == "Square") {
    ... s.position.x ...
    ... s.position.y ...
  } else if(s.tag == "Circle") {
    /*...display for circle…*/
  }
  ...
}
```

B- Delegation

client gives general instructions

each shape knows how to do the task, individually

```
s.displayOn(screen)
```

- each shape is responsible for displaying itself
- each shape (and each position) is responsible for its own representation, invisible from the outside

In the face of change

Suppose there is a new kind of shape

A- Modify client to distinguish new shapes, and implement specific behavior

```
def displayShape(s: Shape): Unit {
  if(s.tag == "Square") {
    ... s.position.x ...
    ... s.position.y ...
  } else if(s.tag == "Circle") {
    /*...display for circle…*/
  }
  ...
}
```

B- Specialize the routine of new shapes.
Client still only says “display yourself”

```
s.displayOn(screen)
```

The OOP paradigm

Perspectives in Software Development Process

Perspective	Description
Conceptual	Represents the concepts in the domain. Little/no regard for the underlying software WHAT AM I RESPONSIBLE FOR?
Specification	Looking at software, but ONLY at the interfaces. No implementation. HOW AM I USED?
Implementation	Code. (*) HOW DO I FULFILL MY RESPONSIBILITIES?

(*) “this is probably the most often-used perspective, but in many ways the specification perspective is often a better one to take”

How perspectives help

"Display yourself"

```
s.displayOn(screen)
```

Client communicates at the **conceptual** level

tell **WHAT** you want, not **HOW** to do it

Shapes are working at the **implementation** level

Big advantage:

Client insulated from changes in implementation & specific behavior as long as concept remains the same

How to think about objects

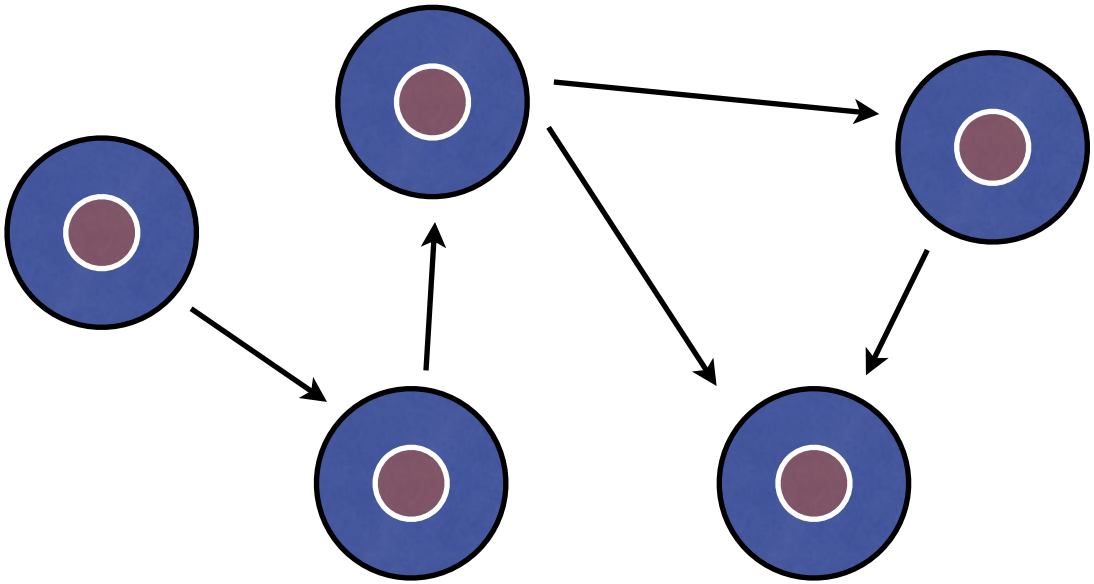
Something with **responsibilities**

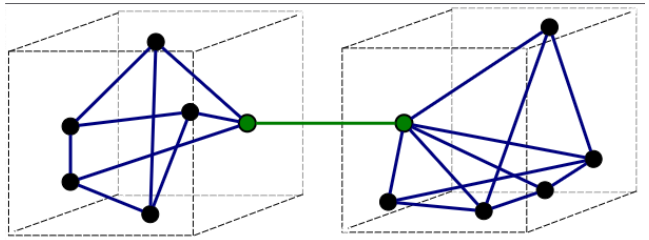
responsible for itself (behavior & structure)

clearly-defined responsibilities

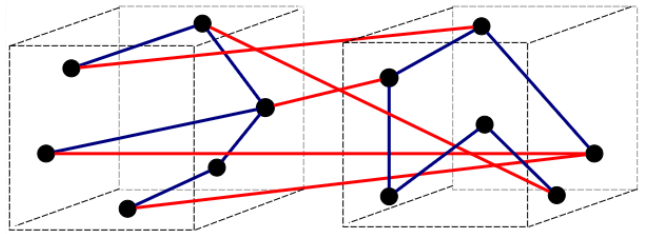
Object	Responsibilities
Client	Know how to get list of shapes Tell shapes to display themselves
Shape	Know how to display itself
DB	Provide the list of shapes

In an OO program,
responsibilities are distributed among many small entities,
with high cohesion
and low coupling





a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

How do we come up with a proper object-oriented decomposition?

As a first approximation:
objects correspond to **nouns** in the domain
methods correspond to **verbs** in the domain

eg. we **display** a **shape**
a **robot** can **talk** and **walk**

OOP with Fowler's Perspectives

Conceptual level

an object is a **set of responsibilities**

Specification level

an object is defined by a **set of methods** that can be invoked (or **messages** that it understands)

Implementation level

an object is **code and data**, and computational **interactions** with others

Object interfaces

Objects have **responsibilities**

Accordingly, they understand some messages / expose some services

Object **interface**

set of methods that others can invoke

aka. “procedural abstraction”

Different languages have different means to support that

Modular object interfaces in Scala

An object can only access other objects through their **public interfaces**

Object interfaces can be defined externally by using traits

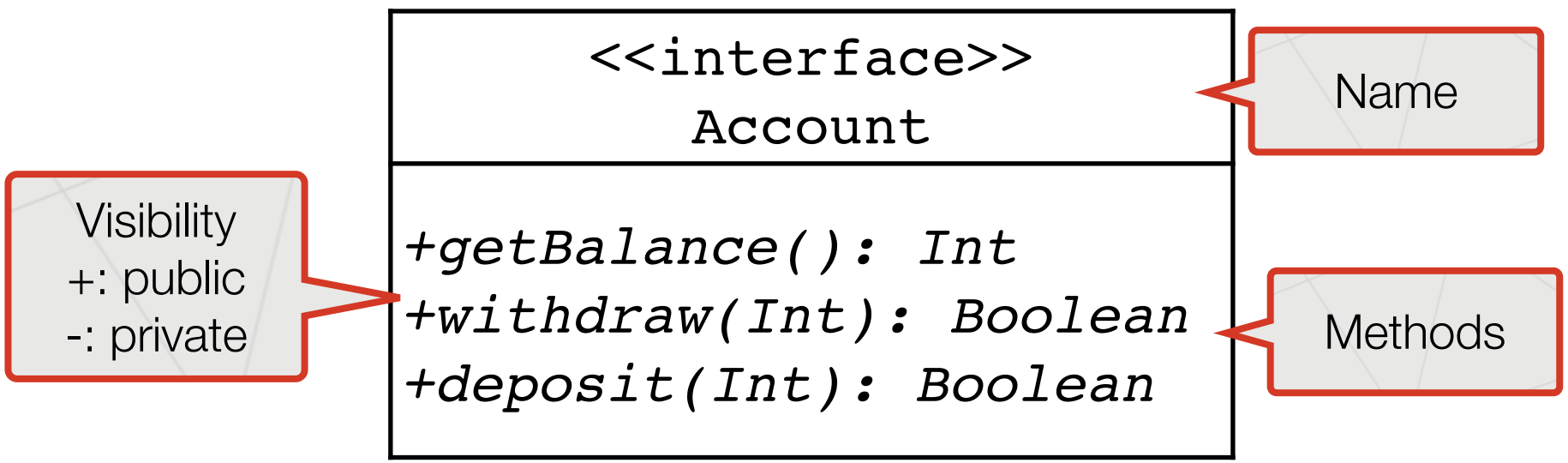
This is a specification of the interface of a bank account

```
trait Account{
  def getBalance(): Int
  def withdraw(n: Int): Boolean
  def deposit(n: Int): Boolean
}
```

We don't have to provide a implementation

This are the methods other object can invoke, or the messages that a bank account understands

Modular object interfaces (UML)



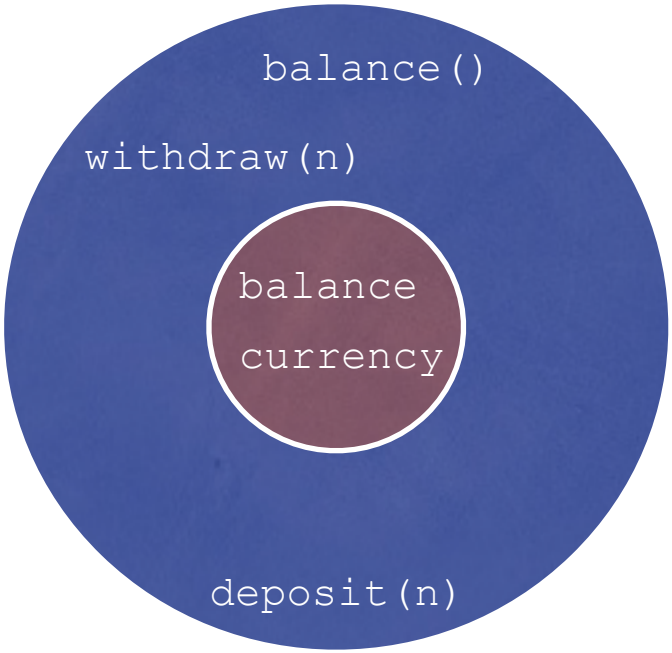
Implementation level

behavior: methods

- public methods form the **object interface**
- may have private methods (*)

state: fields

- aka. instance variables, slots
- may be mutable
- normally NOT visible outside (*)



encapsulation

hide as much as possible

(*) depends on the language

Implementation level

An object can only have **detailed knowledge** about itself

A singleton object is created by using the object keyword

```
object JohnsAccount extends Account{
  /* state: fields */
  private var balance: Int = 10000
  private val currency: String = "CLP"

  /* behavior: methods */
  def getBalance(): Int = balance
  def withdraw(n: Int): Boolean = {
    if(n>=balance){
      balance = balance - n
      true
    } else false
  }
  def deposit(n: Int): Boolean = {
    balance = balance + n
    true
  }
}
```

We can implement an interface defined in a trait using extends

The state are the *private* internals of the object

Now we must implement the methods defined in the trait

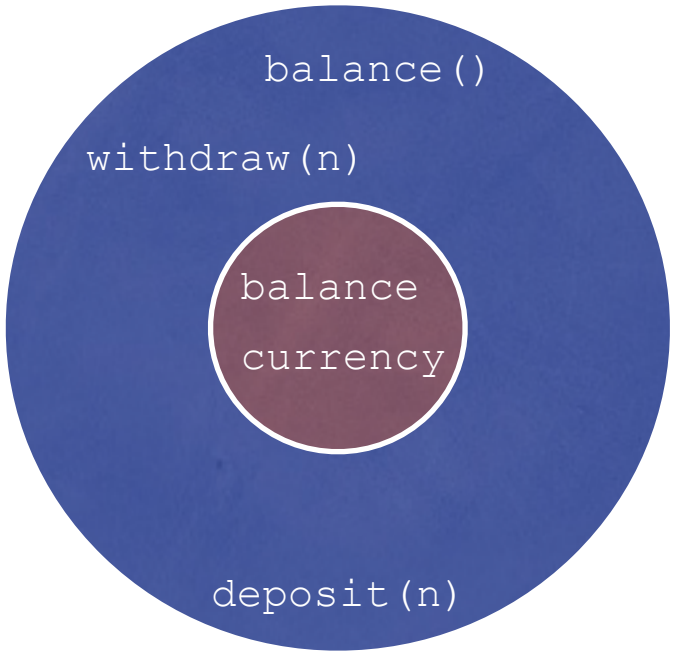
Using objects

Asking an object to do something

"sending a message" or "invoking a method"

can have some effect, can produce a result

```
JohnsAccount.withdraw(50)
```



What is the type of JohnsAccount?

```
object JohnsAccount extends Account{
  /* state: fields */
  private var balance: Int = 10000
  private val currency: String = "CLP"

  /* behavior: methods */
  def getBalance(): Int = balance
  def withdraw(n: Int): Boolean = {
    if(n<=balance){
      balance = balance - n
      true
    } else false
  }
  def deposit(n: Int): Boolean = {
    balance = balance + n
    true
  }
}
```

Account?

Object?

Yes!

But is a good practice to **always use a trait**

Creating objects

In Scala you can create objects literally:

```
object Alexander{
  private val name = "Alexander"
  def talk() = s"${name}: guau!"
}
…
val dog = Alexander
```

No trait: public state and methods form the interface

But how to create other dogs that share methods?

Anonymously:

```
val dog = new {
  private val name = "Alexander"
  def talk() = s"${name}: Ed……ward!"
}
```

Classes: factory of objects

Most languages are class-based

Java, C++, C#, Python, Simula, Smalltalk, Scala, Ruby, etc.

A class is a factory of objects

describes the attributes and methods of the objects it creates

A class can take arguments

No trait (bad practice): a Dog can only talk

```
class Dog(name: String) {
  def talk() = s"${name}: guau!"
}
```

A class can be used as a type, but they are not! (Try to avoid it if possible)

We can use the arguments in the internals

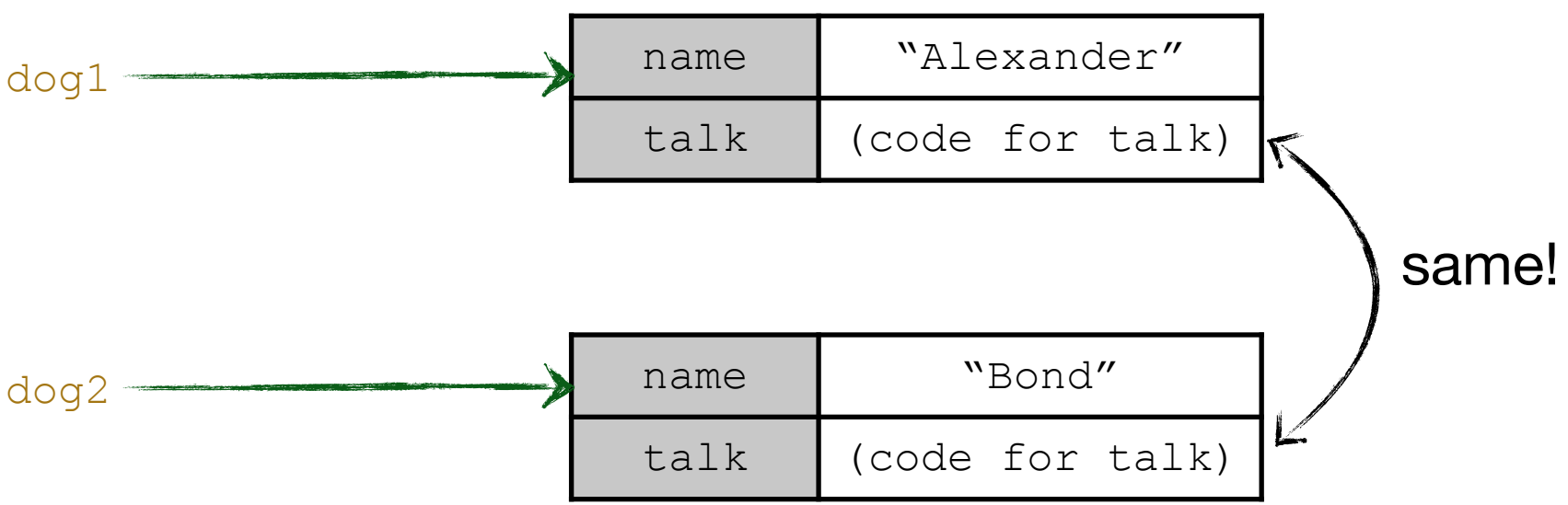
Classes: factory of objects

```
class Dog(name: String) {
  def talk() = s"${name}: guau!"
}
```

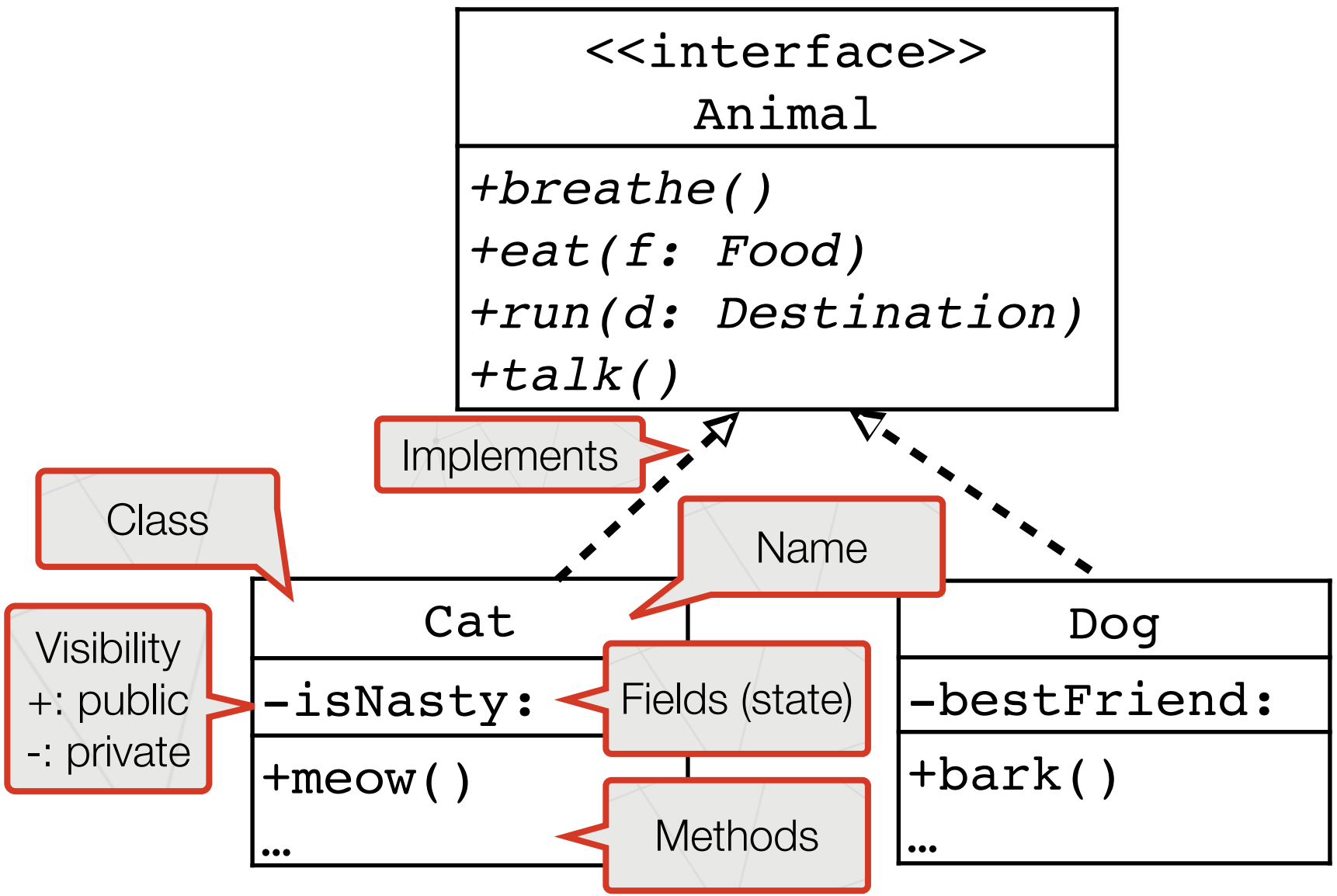
```
val dog1: Dog = new Dog("Alexander")
dog1.talk()

val dog2: Dog = new Dog(“Bond")
dog2.talk()
```

Classes: factory of objects

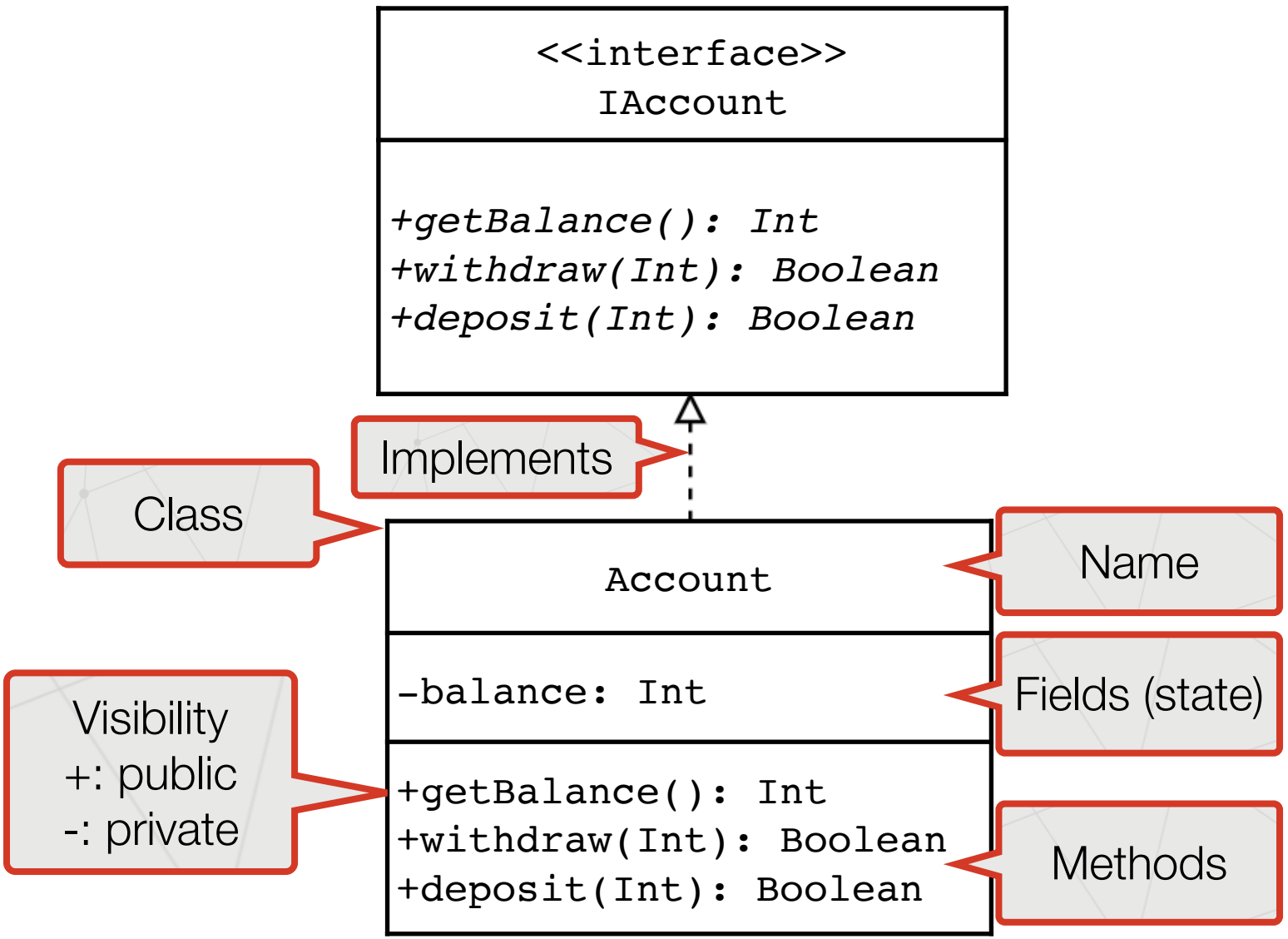
```
val dog1: Dog = new Dog("Alexander")
dog1.talk()

val dog2: Dog = new Dog(“Bond")
dog2.talk()
```



UML diagram example



Back to the account example



Back to the account example

```
trait IAccount{
  def getBalance(): Int
  def withdraw(n: Int): Boolean
  def deposit(n: Int): Boolean
}
class Account(private var balance: Int,
              private val currency: String) extends IAccount{
  /* behavior: methods */
  def getBalance(): Int = balance
  def withdraw(n: Int): Boolean = {
    if(n<=balance){
      balance = balance - n
      true
    } else false
  }
  def deposit(n: Int): Boolean = {
    balance = balance + n
    true
  }
}
```

```
val johnsAccount: IAccount =
          new Account(1000, "CLP")
```

Exercise

Define a class of Point objects

- x and y coordinates
- we need to be able to move a Point

Define a class of Rectangle objects

- a Point as origin, width and height
- we need to be able to move a Rectangle
- we also need to be able to tell its area

Exercise

Now, make a small demo

create a point, two rectangles

print the area of the rectangles

move one of them

(add a way to represent points and rectangle as strings and use that to "display" the figures)

License

Diapositivas basadas en Éric Tanter

dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

/ DCCUCHILE