Essential of Object Oriented Programming

Alexandre Bergel
http://bergel.eu
02/09/2020

Goal of this lecture

This lecture will essentially be a *Java introduction*

Emphasis on *what an object is*

Highlight *some important* particularities of Java

Outline

1. Java by example
    1. Small illustrative scenario with an artificial neuron
    2. Extending Neuron into ReluNeuron
2. Class inheritance
3. Terminology

Outline

1. Java refresher

  1. Small illustrative scenario with an artificial neuron

  2. Extending Neuron into ReluNeuron

2. Class inheritance

3. Terminology

Defining a Neuron as a first example...

Deep learning is about network of artificial neurons

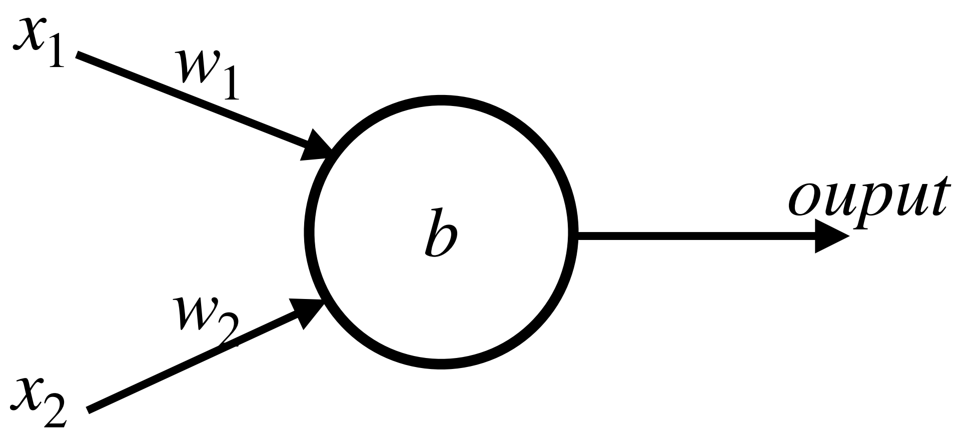
We will *model an artificial neuron* as a first example

In our model, *a neuron is a simple machine* that can make decision

Modeling a neuron is simply *an entertaining example*. It does not contribute to the content of the lecture

Defining a Neuron as a first example...

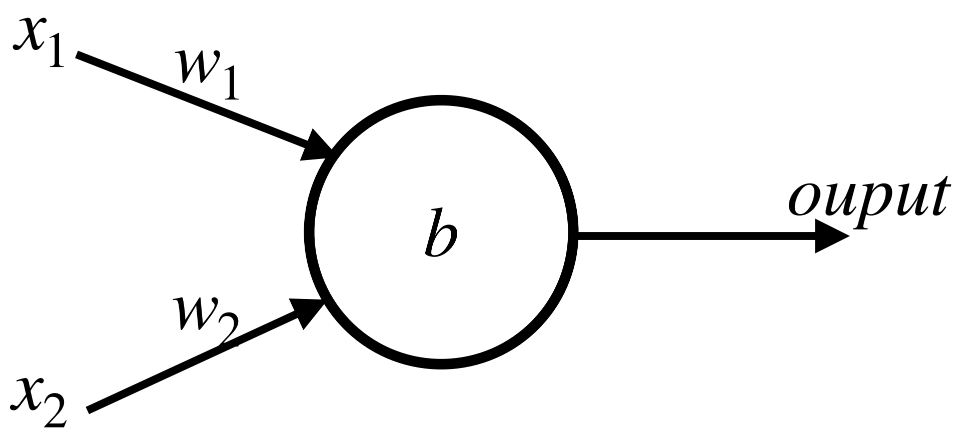
A neuron has many inputs.
We will only consider 2 inputs for now



$$output = 0 \text{ if } w_1 * x_1 + w_2 * x_2 + b \leq 0$$
$$output = 1 \text{ if } w_1 * x_1 + w_2 * x_2 + b > 0$$

Defining a Neuron as a first example...

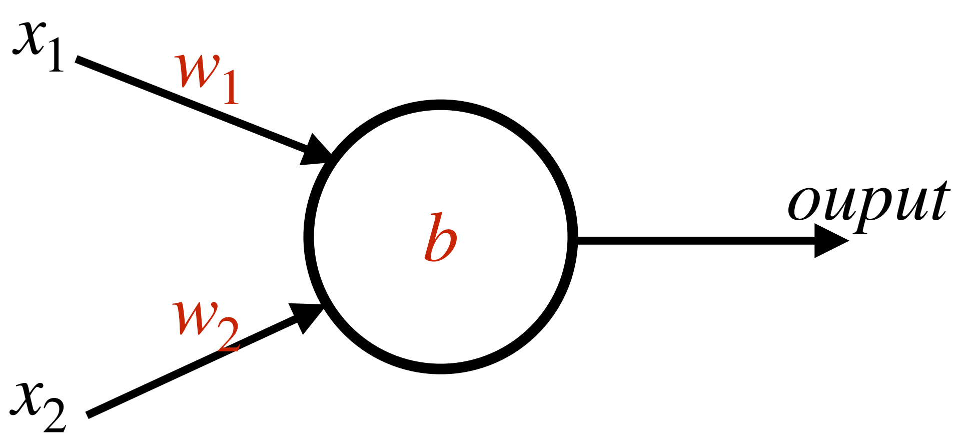
A neuron has many inputs.
We will only consider 2 inputs for now



We call this value z

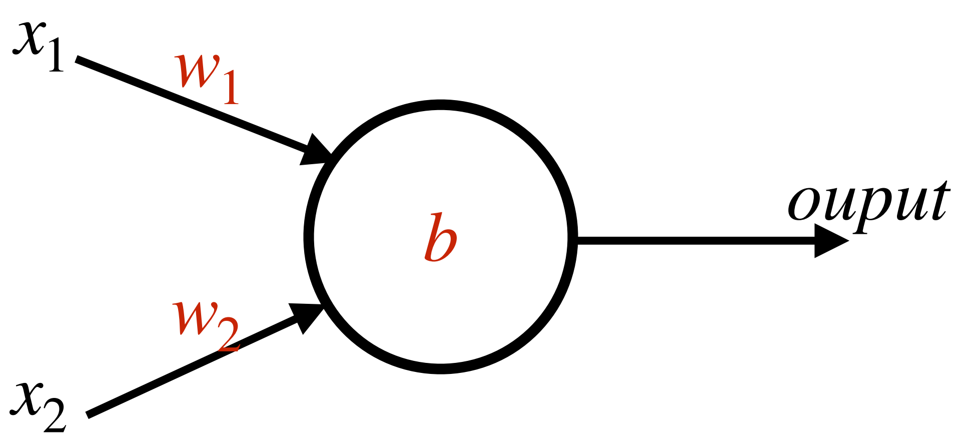
$$output = 0 \text{ if } \boxed{w_1 * x_1 + w_2 * x_2 + b} \leq 0$$
$$output = 1 \text{ if } \boxed{w_1 * x_1 + w_2 * x_2 + b} > 0$$

Defining a Neuron as a first example...



The class Neuron will therefore define 3 variables

Defining a Neuron as a first example...



The class Neuron will therefore define 3 variables

A neuron can answer 2 different messages

- `computeZ`: compute the intermediary Z value
- `feed`: which returns the output value

Definition of Neuron Class

```
package c3002.artificial.neuron;

public class Neuron {
    private double weight1, weight2;
    private double bias;
   …
```

Definition of Neuron Class

```
…
public Neuron(double w1, double w2, double b) {
    weight1 = w1;
    weight2 = w2;
    bias = b;
}
…
```

Definition of Neuron Class

```
…
public double computeZ(double input1, double input2) {
    return input1 * weight1 +
           input2 * weight2 + bias;
}
…
```

Definition of Neuron Class

```
…
  public double feed(double input1, double input2) {
          double z = this.computeZ(input1, input2);
          if(z <= 0)
              return 0;
          else
              return 1;
    }
}
```

Definition of NeuronExample

```java
package c3002.artificial.neuron;

public class NeuronExample {
    public static void main(String[] args){
        Neuron or = new Neuron(1.0, 1.0, -0.5);
        System.out.println("0 OR 0 = " + or.feed(0, 0));
        System.out.println("1 OR 0 = " + or.feed(1, 0));
    }
}
```

Definition of NeuronExample

Entry point of the program

```
package c3002.artificial.neur

public class NeuronExample {
    public static void main(String[] args){
        Neuron or = new Neuron(1.0, 1.0, -0.5);
        System.out.println("0 OR 0 = " + or.feed(0, 0));
        System.out.println("1 OR 0 = " + or.feed(1, 0));
    }
}
```

Definition of NeuronExample

Creation of a
Neuron object / instance

```java
package c3002.artificial.neur

public class NeuronExample {
    public static void main(String[] args){
        Neuron or = new Neuron(1.0, 1.0, -0.5);
        System.out.println("0 OR 0 = " + or.feed(0, 0));
        System.out.println("1 OR 0 = " + or.feed(1, 0));
    }
}
```

Definition of NeuronExample

Compute the output of a neuron

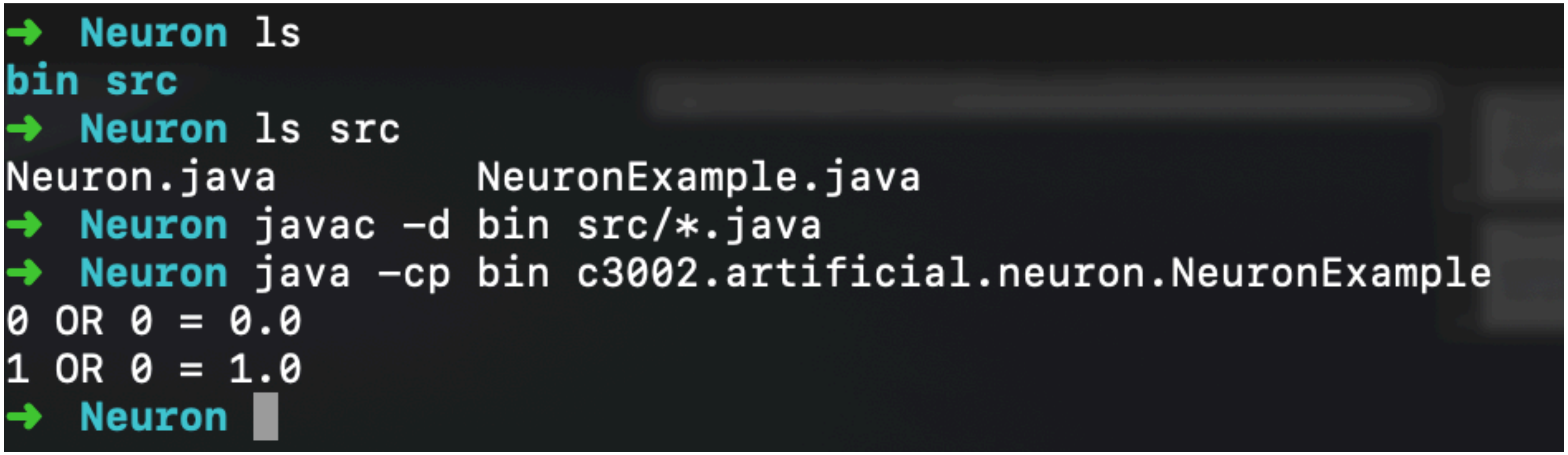
```java
package c3002.artificial.neur

public class NeuronExample {
    public static void main(String[] args){
        Neuron or = new Neuron(1.0, 1.0, -0.5);
        System.out.println("0 OR 0 = " + or.feed(0, 0));
        System.out.println("1 OR 0 = " + or.feed(1, 0));
    }
}
```

Definition of NeuronExample

Convert the double into a string, and concatenate

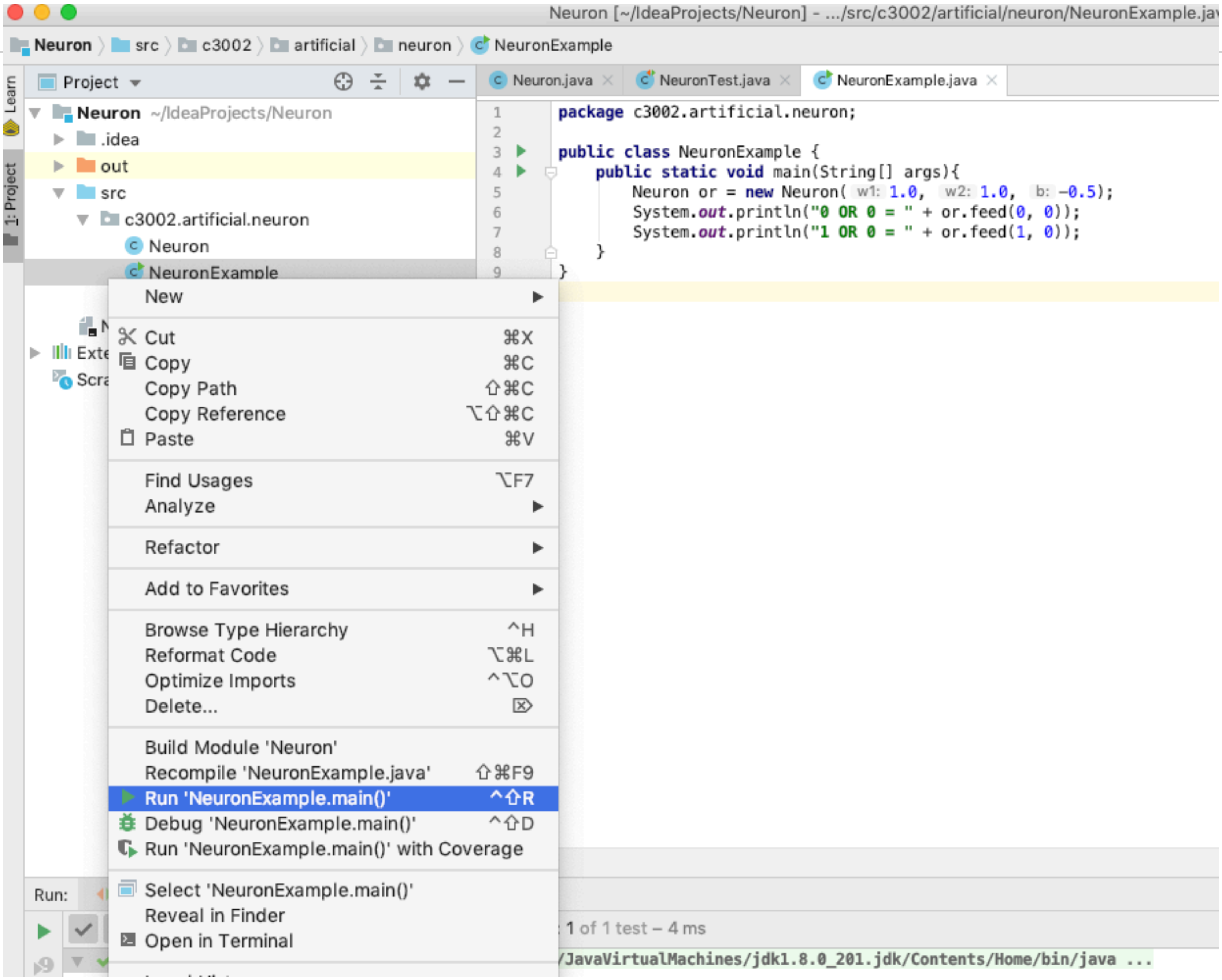
```
package c3002.artificial.neur

public class NeuronExample {
    public static void main(String[] args){
        Neuron or = new Neuron(1.0, 1.0, -0.5);
        System.out.println("0 OR 0 = " + or.feed(0, 0));
        System.out.println("1 OR 0 = " + or.feed(1, 0));
    }
}
```

Running the example (using the command line)



```
➜  Neuron ls
bin src
➜  Neuron ls src
Neuron.java          NeuronExample.java
➜  Neuron javac -d bin src/*.java
➜  Neuron java -cp bin c3002.artificial.neuron.NeuronExample
0 OR 0 = 0.0
1 OR 0 = 1.0
➜  Neuron 
```

Running the example (using IntelliJ)



Neuron [~/IdeaProjects/Neuron] - .../src/c3002/artificial/neuron/NeuronExample.ja

```java
package c3002.artificial.neuron;

public class NeuronExample {
    public static void main(String[] args){
        Neuron or = new Neuron( w1: 1.0,  w2: 1.0,  b: -0.5);
        System.out.println("0 OR 0 = " + or.feed(0, 0));
        System.out.println("1 OR 0 = " + or.feed(1, 0));
    }
}
```

New	▶
Cut	⌘X
Copy	⌘C
Copy Path	⇧⌘C
Copy Reference	⌥⇧⌘C
Paste	⌘V
Find Usages	⌥F7
Analyze	▶
Refactor	▶
Add to Favorites	▶
Browse Type Hierarchy	^H
Reformat Code	⌥⌘L
Optimize Imports	^⌥O
Delete...	⌦
Build Module 'Neuron'	
Recompile 'NeuronExample.java'	⇧⌘F9
Run 'NeuronExample.main()'	^⇧R
Debug 'NeuronExample.main()'	^⇧D
Run 'NeuronExample.main()' with Coverage	
Select 'NeuronExample.main()'	
Reveal in Finder	
Open in Terminal	

```
/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
```

Some of the important parts that you should not have missed! ...

- `Neuron` knows what `or` looks like and how it behaves
  - `Neuron` knows how to interpret the orders given to `or`
- The `or` object only knows
  - the value of `weight1, weight2`, and `bias`
  - who created it

Some of the important parts that you should not have missed!

`NeuronExample` *sends* to `or` some orders, defined in term of *messages*

`NeuronExample` *cannot send a message* to `or` that is not understood

- In Python, JavaScript, or Ruby, one can send a message that is not understood
- In Java or C#, messages are always understood

Java particularities

Java is a *class-based object-oriented language*

... but not completely

- a *class instantiation is not done through message sending*, but with an *operator*
- Java contains primitive types, which are not objects

Static methods are not looked up

- only methods (or also called instance methods) that are not private are looked up
- we will come back on that point in the future
- E.g., the main() method is called directly by the VM, without instantiating the class `NeuronExample`

Defining a different kind of Neuron…

A Relu Neuron is a different kind of neuron.

The same Z value is computed, but the output is slightly different

Note that the `ReluNeuron` cannot be used to have the `or` behavior we defined earlier

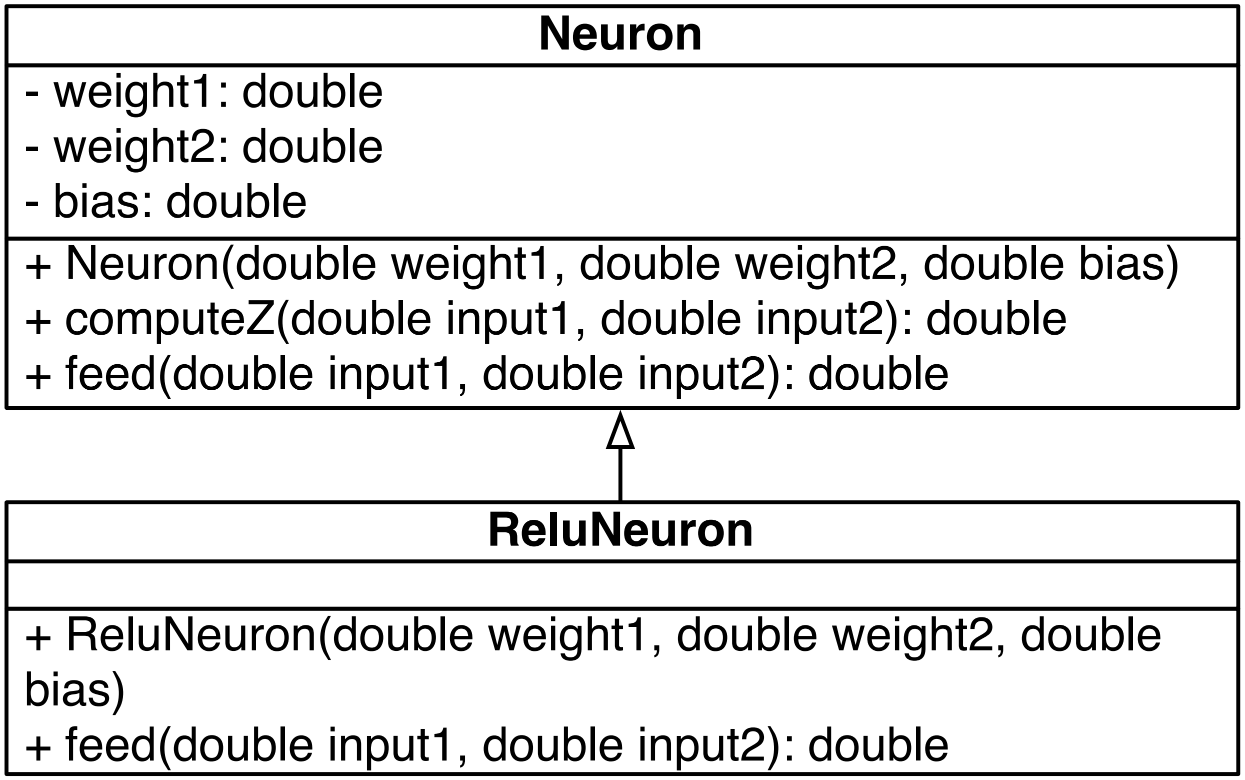
Defining a different kind of Neuron…

```java
package c3002.artificial.neuron;

public class ReluNeuron extends Neuron {
    public ReluNeuron(double w1, double w2, double b) {
        super(w1, w2, b);
    }

    public double feed(double input1, double input2) {
        double z = this.computeZ(input1, input2);
        return (z > 0) ? z : 0;
    }
}
```

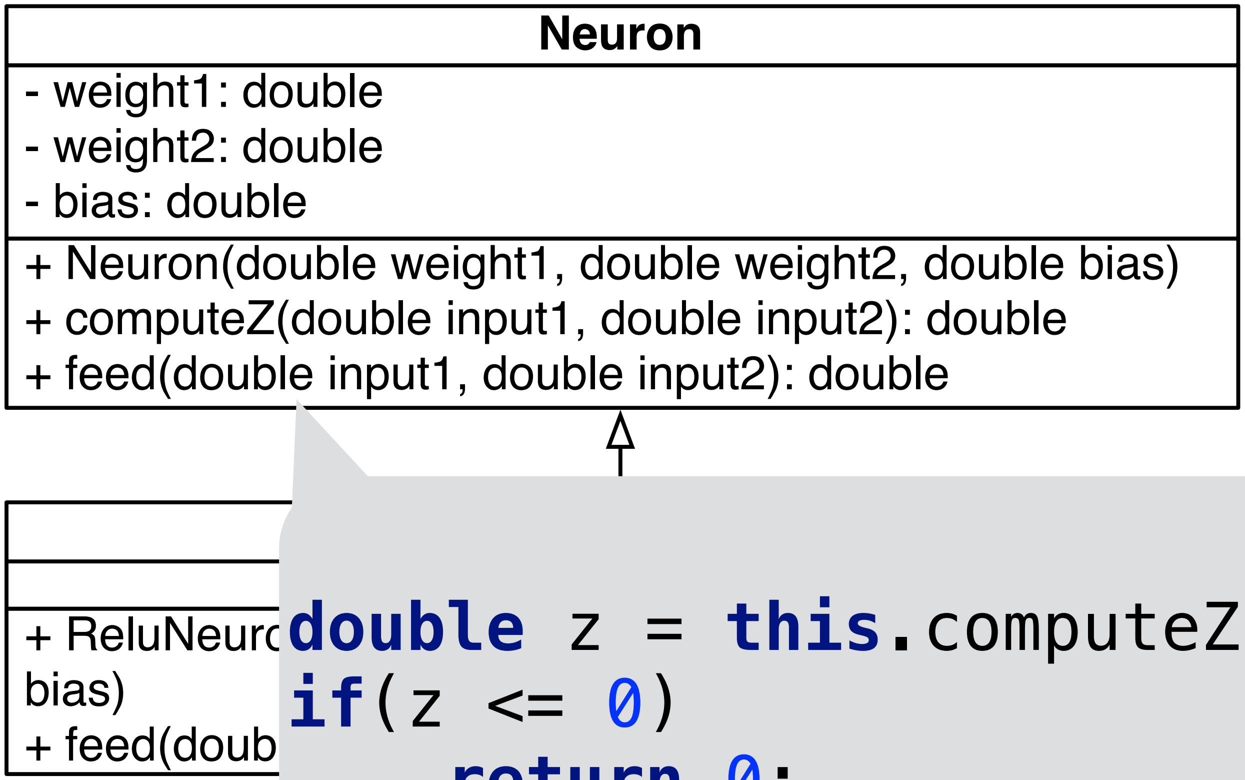
Outline

1. Java refresher
    1. Small illustrative scenario with the class Point
    2. Extending Point into PositivePoint
2. **Class inheritance**
3. Terminology

Class inheritance

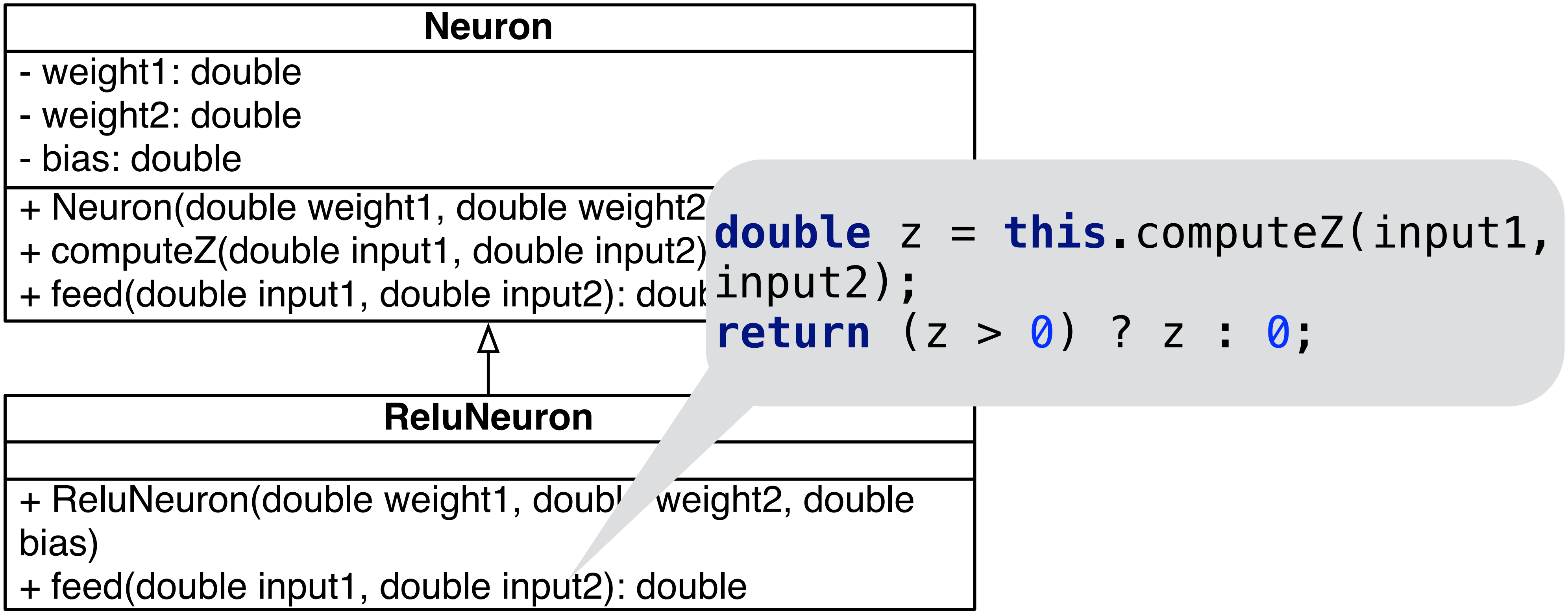

Neuron
- weight1: double
- weight2: double
- bias: double
+ Neuron(double weight1, double weight2, double bias)
+ computeZ(double input1, double input2): double
+ feed(double input1, double input2): double
ReluNeuron
+ ReluNeuron(double weight1, double weight2, double bias)
+ feed(double input1, double input2): double

Class inheritance


Neuron
- weight1: double
- weight2: double
- bias: double
+ Neuron(double weight1, double weight2, double bias)
+ computeZ(double input1, double input2): double
+ feed(double input1, double input2): double
+ ReluNeuro
bias)
+ feed(doub

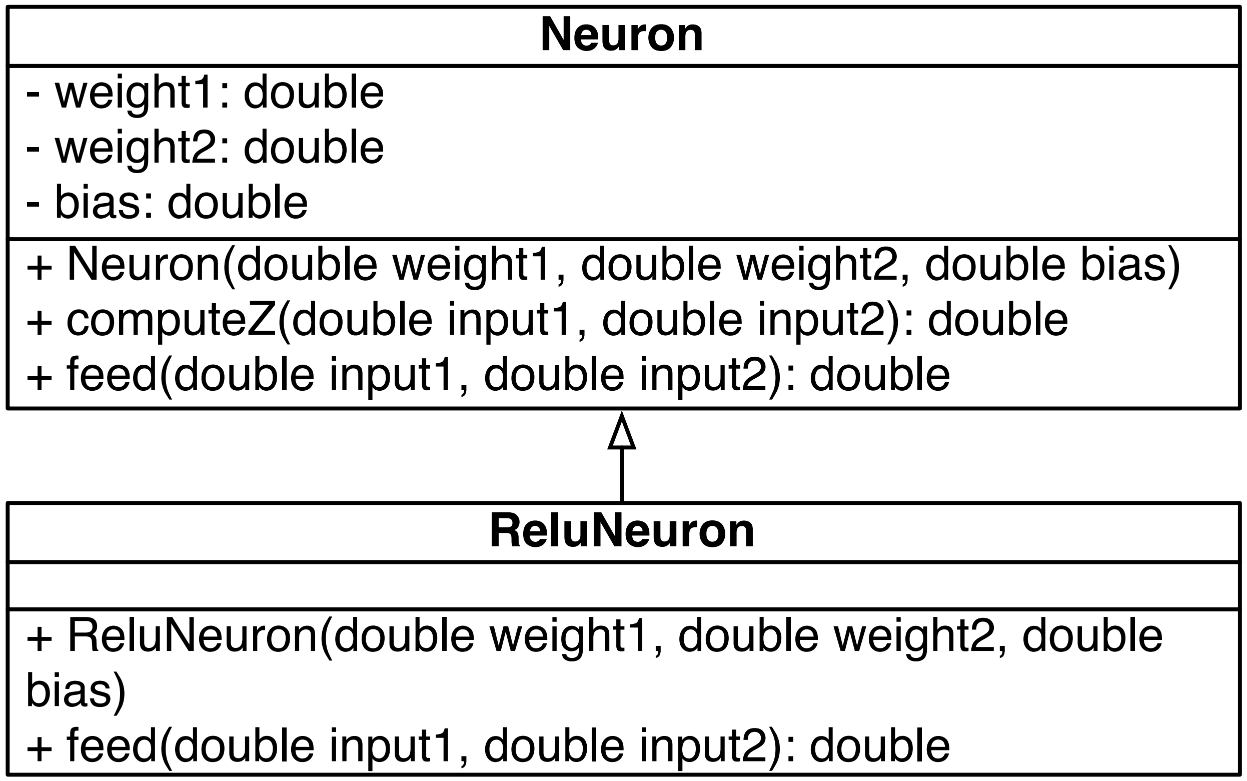
```
double z = this.computeZ(input1, input2);
if(z <= 0)
    return 0;
else
    return 1;
```

Class inheritance


Neuron
- weight1: double
- weight2: double
- bias: double
+ Neuron(double weight1, double weight2
+ computeZ(double input1, double input2)
+ feed(double input1, double input2): doub
ReluNeuron
+ ReluNeuron(double weight1, doubl weight2, double bias)
+ feed(double input1, double input2): double

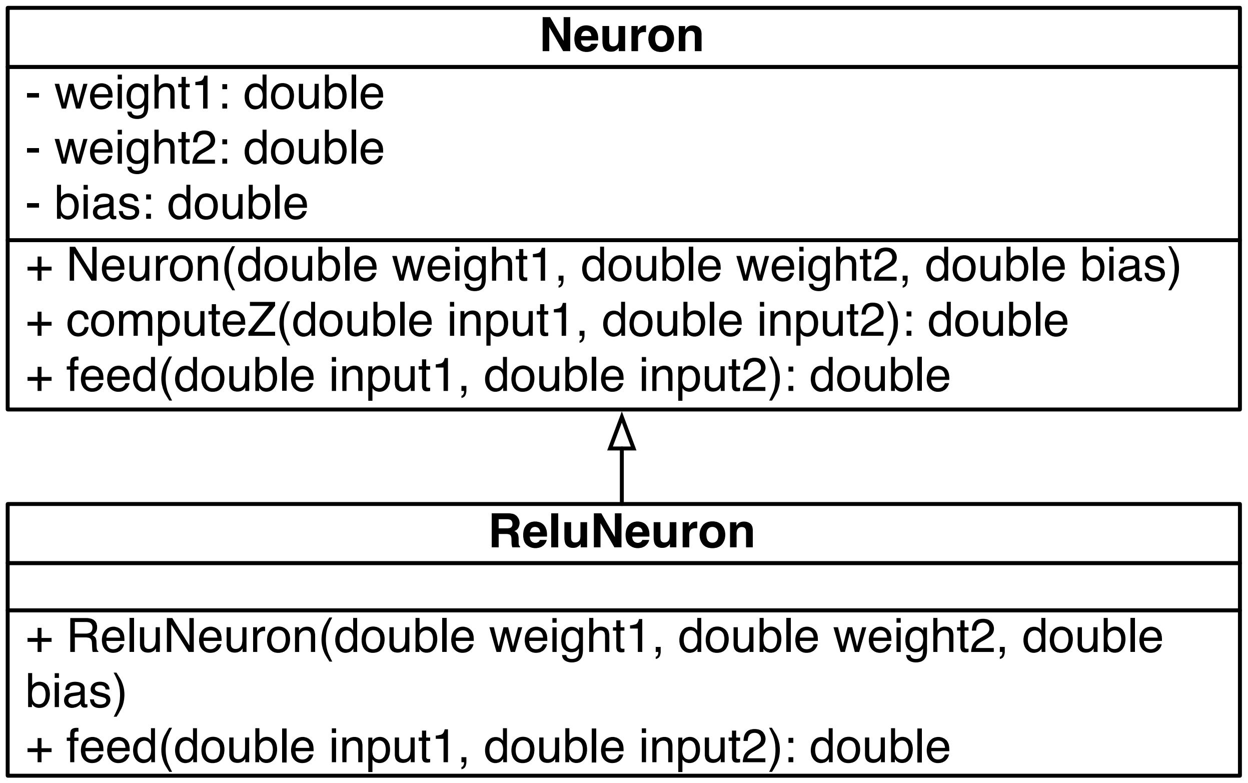
```java
double z = this.computeZ(input1,
input2);
return (z > 0) ? z : 0;
```

Class inheritance


Neuron
- weight1: double
- weight2: double
- bias: double
+ Neuron(double weight1, double weight2, double bias)
+ computeZ(double input1, double input2): double
+ feed(double input1, double input2): double
ReluNeuron
+ ReluNeuron(double weight1, double weight2, double bias)
+ feed(double input1, double input2): double

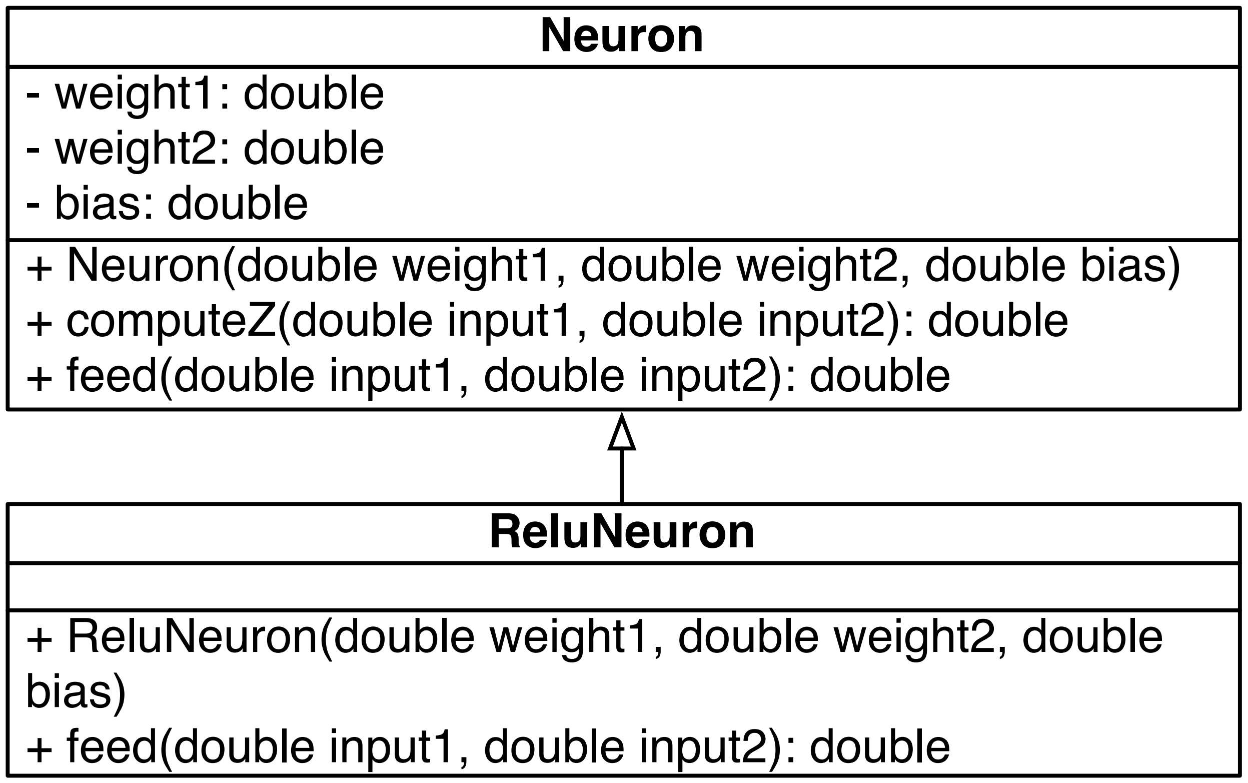
```
Neuron n = new Neuron(1,1,-0.5);
n.feed(1,1)
=> execute Neuron.feed(…)
```

Class inheritance


Neuron
- weight1: double
- weight2: double
- bias: double
+ Neuron(double weight1, double weight2, double bias)
+ computeZ(double input1, double input2): double
+ feed(double input1, double input2): double
ReluNeuron
+ ReluNeuron(double weight1, double weight2, double bias)
+ feed(double input1, double input2): double

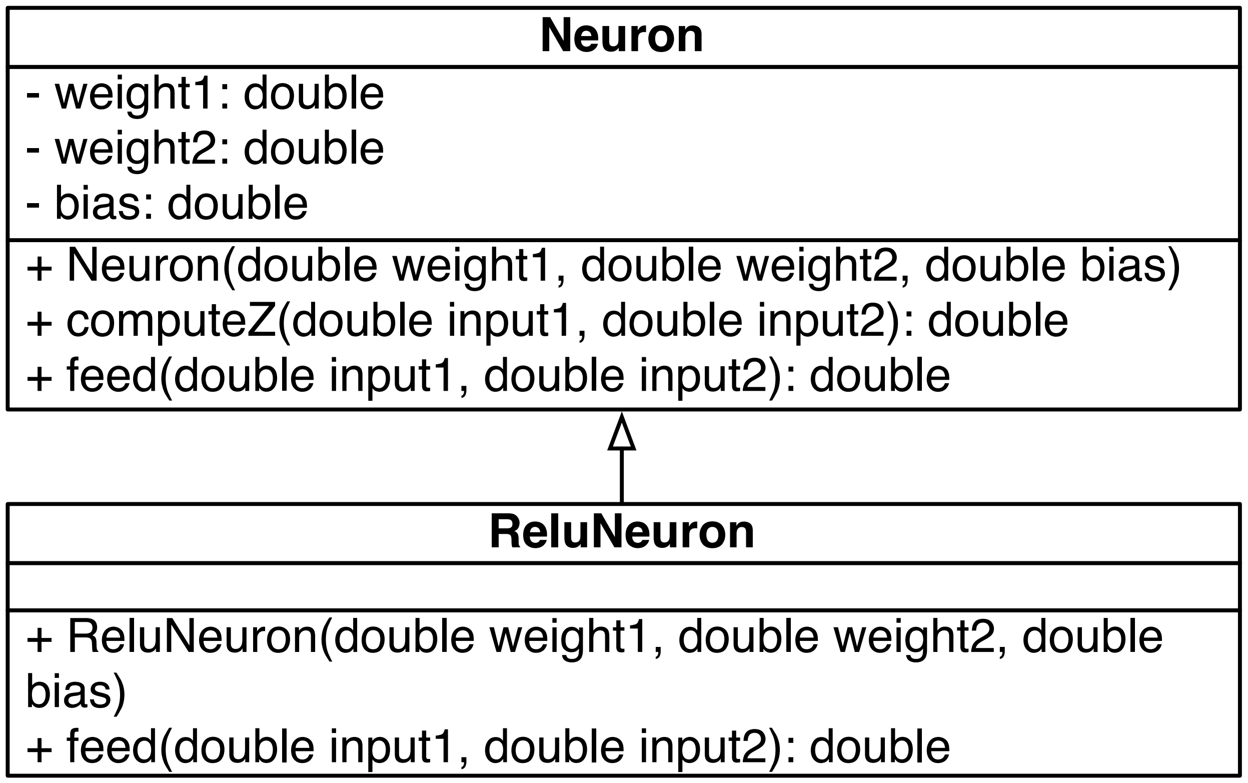
```
ReluNeuron n = new ReluNeuron(1,1,-0.5);
n.feed(1,1)
=> execute ReluNeuron.feed(…)
```

Class inheritance


Neuron
- weight1: double
- weight2: double
- bias: double
+ Neuron(double weight1, double weight2, double bias)
+ computeZ(double input1, double input2): double
+ feed(double input1, double input2): double
ReluNeuron
+ ReluNeuron(double weight1, double weight2, double bias)
+ feed(double input1, double input2): double

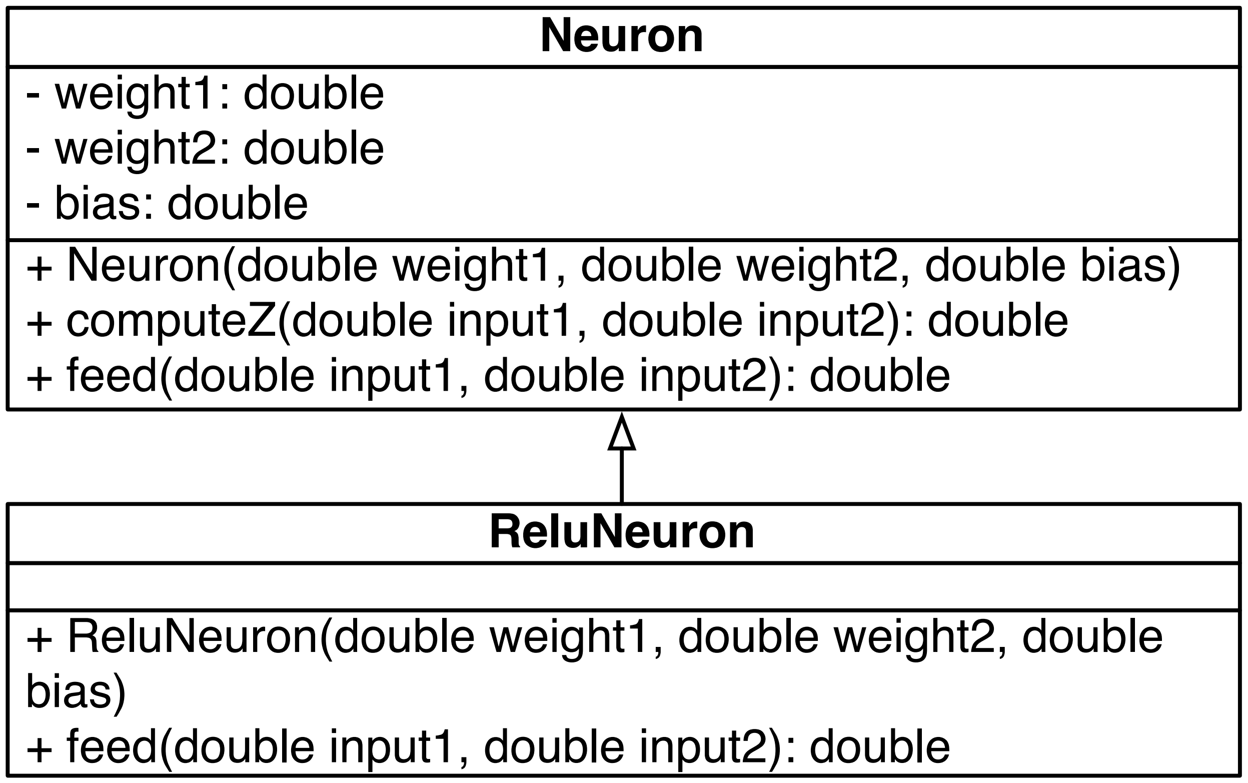
```
Neuron n = new ReluNeuron(1,1,-0.5);
n.computeZ(1,1)
=> execute Neuron.computeZ(…)
```

Class inheritance


Neuron
- weight1: double
- weight2: double
- bias: double
+ Neuron(double weight1, double weight2, double bias)
+ computeZ(double input1, double input2): double
+ feed(double input1, double input2): double
ReluNeuron
+ ReluNeuron(double weight1, double weight2, double bias)
+ feed(double input1, double input2): double

```
Neuron n = new ReluNeuron(1,1,-0.5);
n.feed(1,1)
=> ??
```

Class inheritance



Neuron
- weight1: double
- weight2: double
- bias: double
+ Neuron(double weight1, double weight2, double bias)
+ computeZ(double input1, double input2): double
+ feed(double input1, double input2): double
ReluNeuron
+ ReluNeuron(double weight1, double weight2, double bias)
+ feed(double input1, double input2): double

```
Neuron n = new ReluNeuron(1,1,-0.5);
n.feed(1,1)
=> execute ReluNeuron.feed(…)
```

Class inheritance

- During the first weeks of the semester we will explain *how* inheritance works
- However, understand *when* to use inheritance is the topic of the whole semester
- Class inheritance is highly powerful:
  - It may bring fantastic property regarding extensibility in a software system
  - But it may be devastating if not properly used

Outline

1. Java refresher
    1. Small illustrative scenario with the class Point
    2. Extending Point into PositivePoint
2. Class inheritance
3. **Terminology**

Terminology

Object

"An object is a software machine allowing programs to access and modify a collection of data" -- *Class of Touch, Bertrand Meyer*

"Objects are not just simple bundles of logic and data. They are responsible members of an object community" -- *Object Design, Rebecca Wirfs-Brock and Alan McKean*

An object has a *unique position in memory*, often assimilated as its identity

An object knows from which class it has been created from (for class-based object-oriented programming languages like Java, C#, Smalltalk)

An object *understands* the *messages* for the methods inherited and defined in its class

Terminology

Class

A *class is primarily an object factory*

It is defined as a set of variable declarations and method definitions

Conceptually: *class = name + variables + methods + superclass*

In Java: *class = name + variables + methods + superclass + interfaces + static methods + ...*

Terminology

Method

Executable piece of code

A method execution ends (i) when no more instruction has to be executed; (ii) when a return statement is reached; (ii) when an exception is raised

The control flow is returned to its caller method when the method return

Can access to the *this* and *super* pseudo-variables (only in instance method; cannot be used in a static method in Java)

Terminology

Inheritance / subclasses

relation of *specialization* between classes

a subclass *inherits attributes* and *behavior* from its superclass

it is considered *bad programming style* to use inheritance for code *reuse only*

Terminology

Polymorphism

is the ability of one type A to appear as and be used like another type B

polymorphism plays a key difference between message sending and function invocation

```
Neuron n = new ReluNeuron(1,1,-0.5);
```

What you should know!

What is the difference between an *object* and *class*?

What is the difference between *class reuse* and *class specialization*?

What is a *constructor*?

The difference between *function invocation* and *sending messages*

Can you answer these questions?

Why do objects *“send messages”* instead of *“executing methods”*?

Can you imagine an object model in which a *class is also an object*?

Why *polymorphism* and *class inheritance* are so tightly related in Java?

License

dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

/ DCCUCHILE