COMP201 – Software Engineering I
Lecture 28 - Implementation

Lecturer: Dr T Carroll

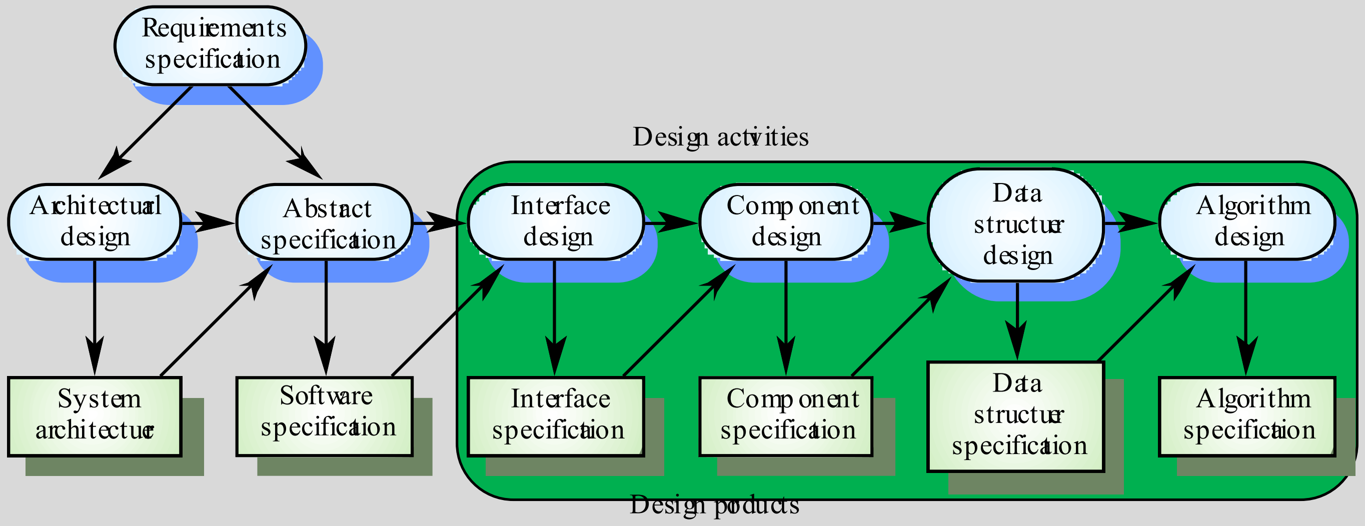
Email: Thomas.Carroll2@Liverpool.ac.uk

Office: G.14

See Vital for all notes

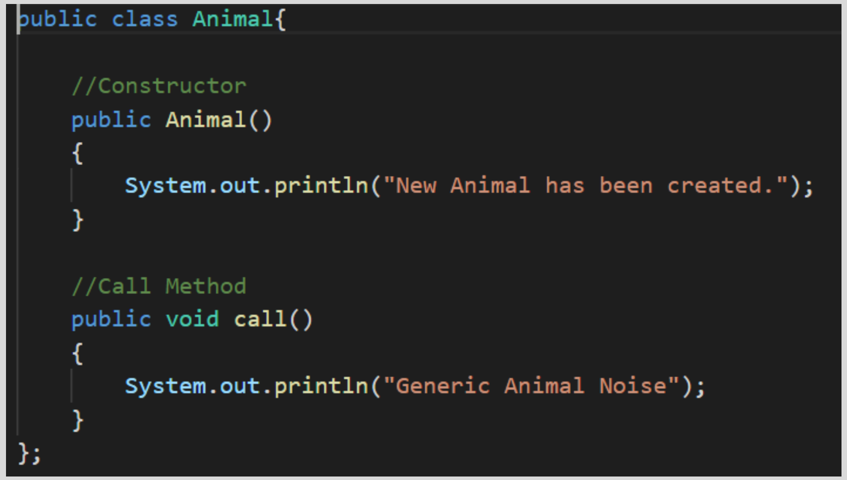
The Software Life Cycle

- Generic Processes cover the software lifecycle
- Each process model has components that cover:
  - Requirements Engineering
  - Software Specification
  - Software Design
  - Implementation
  - Testing
  - Integration
  - Installation

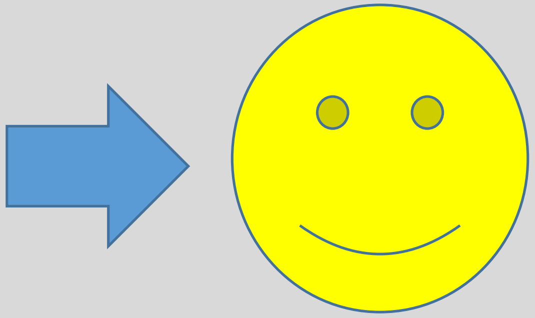


Implementation

- Takes the **design and specification** to **reality**
- Translates **diagrams, pseudocode** and **formal specifications** into **executable code**



```
public class Animal{

    //Constructor
    public Animal()
    {
        System.out.println("New Animal has been created.");
    }

    //Call Method
    public void call()
    {
        System.out.println("Generic Animal Noise");
    }
};
```



HOW can we do this?

Coming Up…

Coming Up…

- Class Diagrams to Java Classes
- Java Inheritance
- Java Abstract Classes
- Java Interfaces
- Java Packages

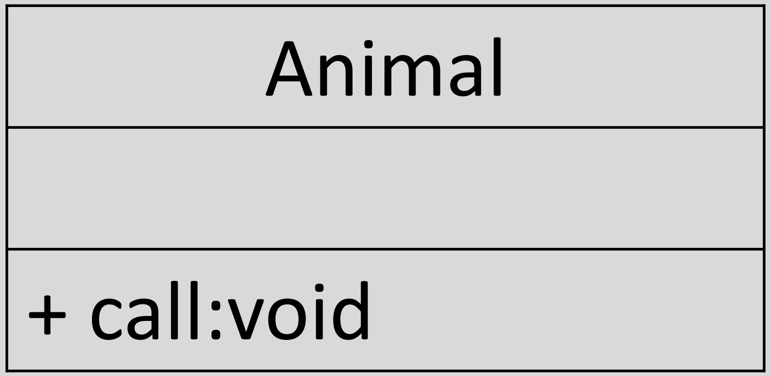
We wont be using an IDE here…. Just your favourite text editor and the terminal!

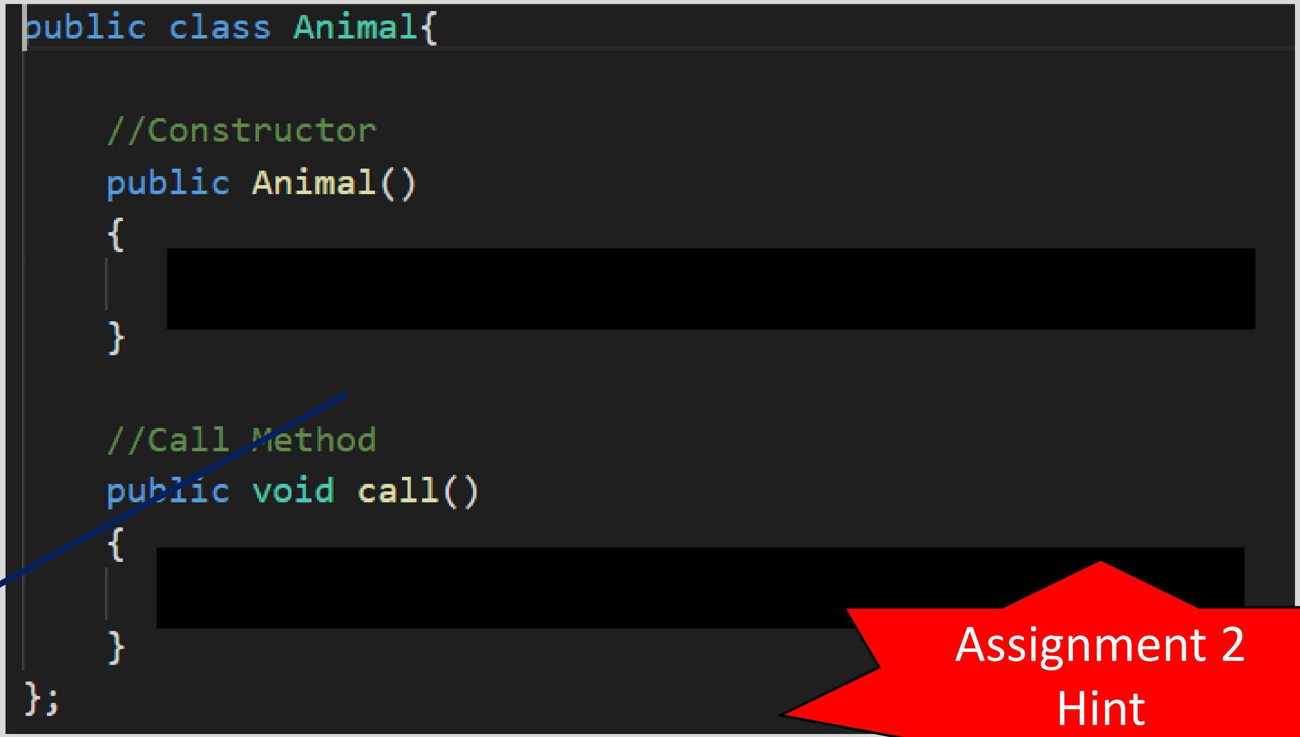
See Software Engineering Tools COMP285 for more in depth study….

Class Diagrams to Classes

UML Class Diagrams

- UML Class Diagrams can be **directly mapped** to an OO Program
- Class diagrams **don't** tell us much about the **behaviour** of the program



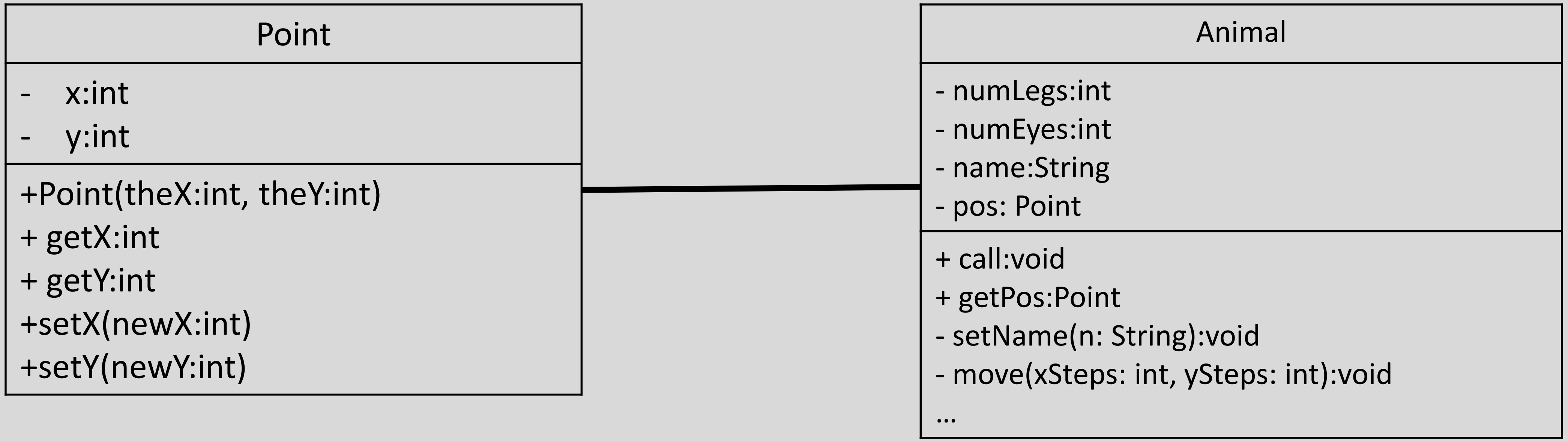


```
public class Animal{

    //Constructor
    public Animal()
    {
        [redacted]
    }

    //Call Method
    public void call()
    {
        [redacted]
    }
};
```

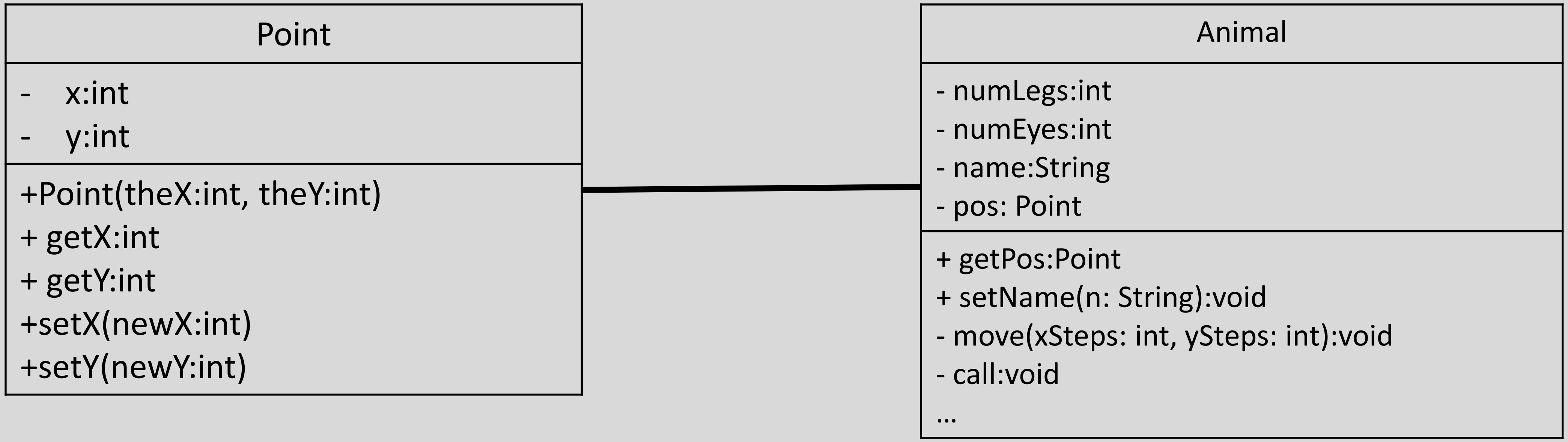
Skeleton class,
Method stubs

Assignment 2
Hint

TASK: Create these as Java Skeleton classes

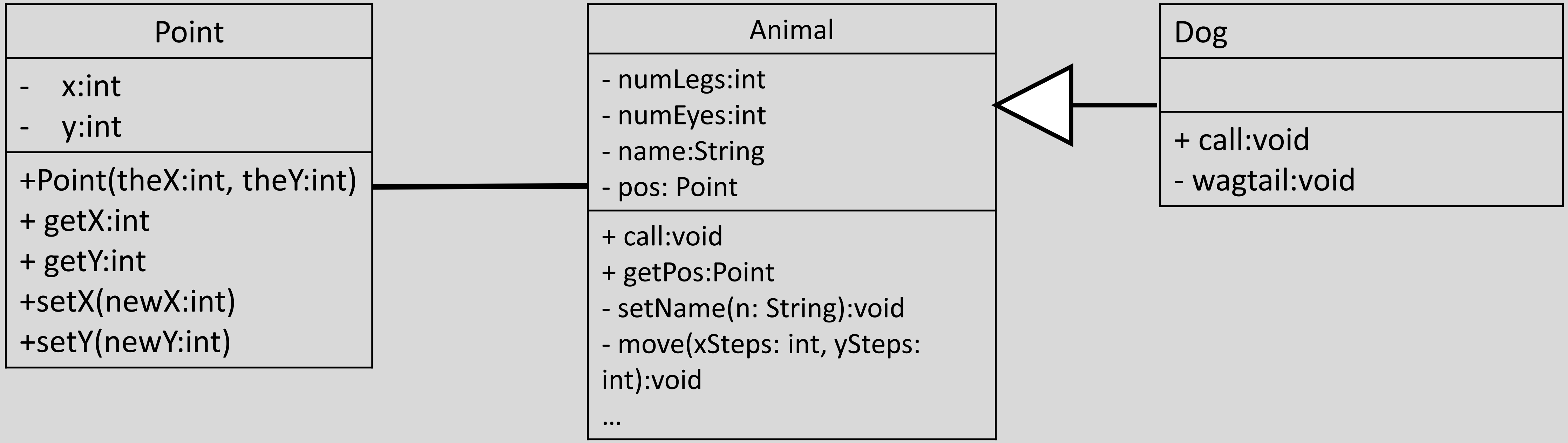


Fun Extra Info: How does this look in C++?



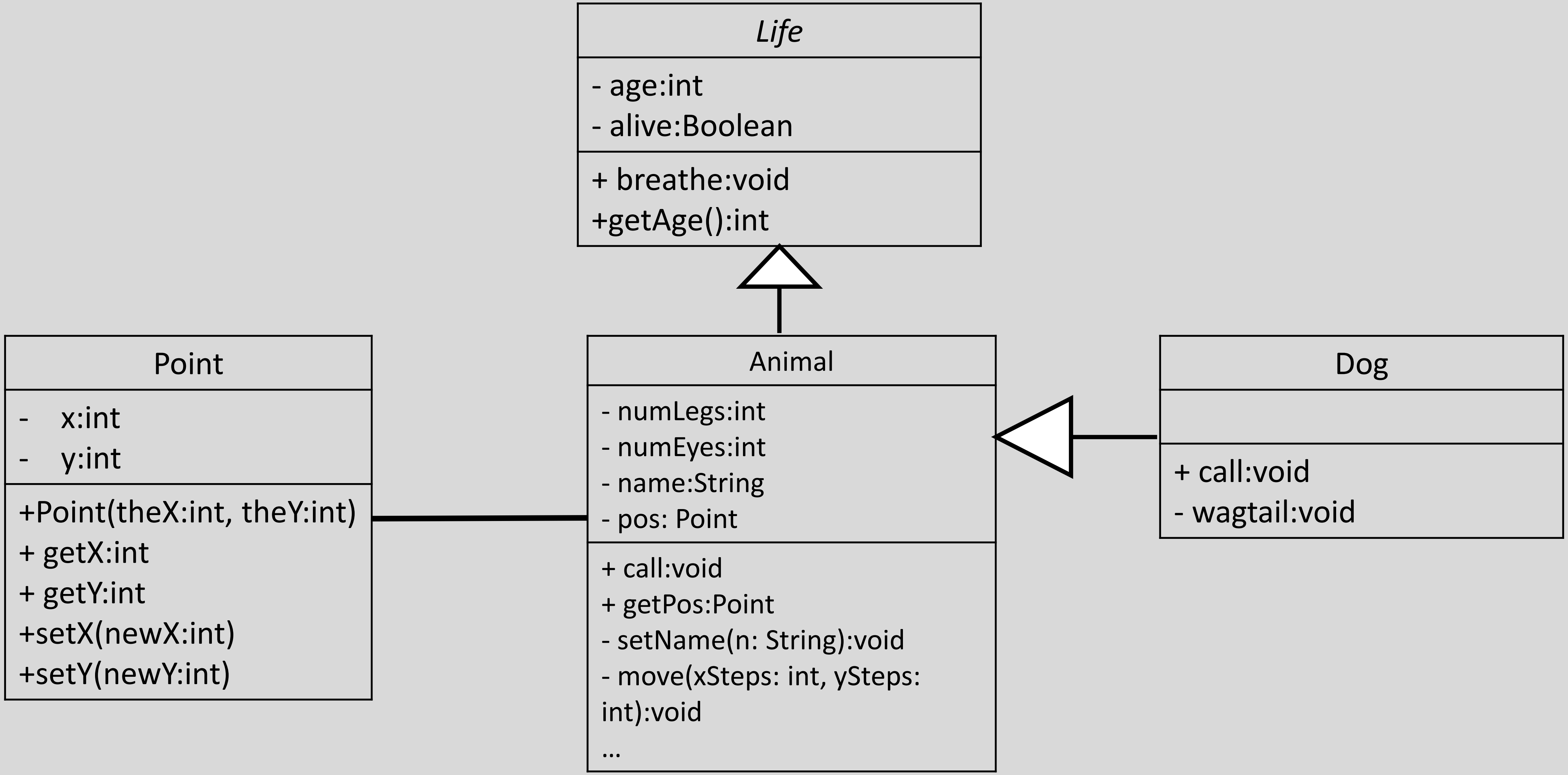
Inheritance

TASK: Extend your previous Java Skeleton classes



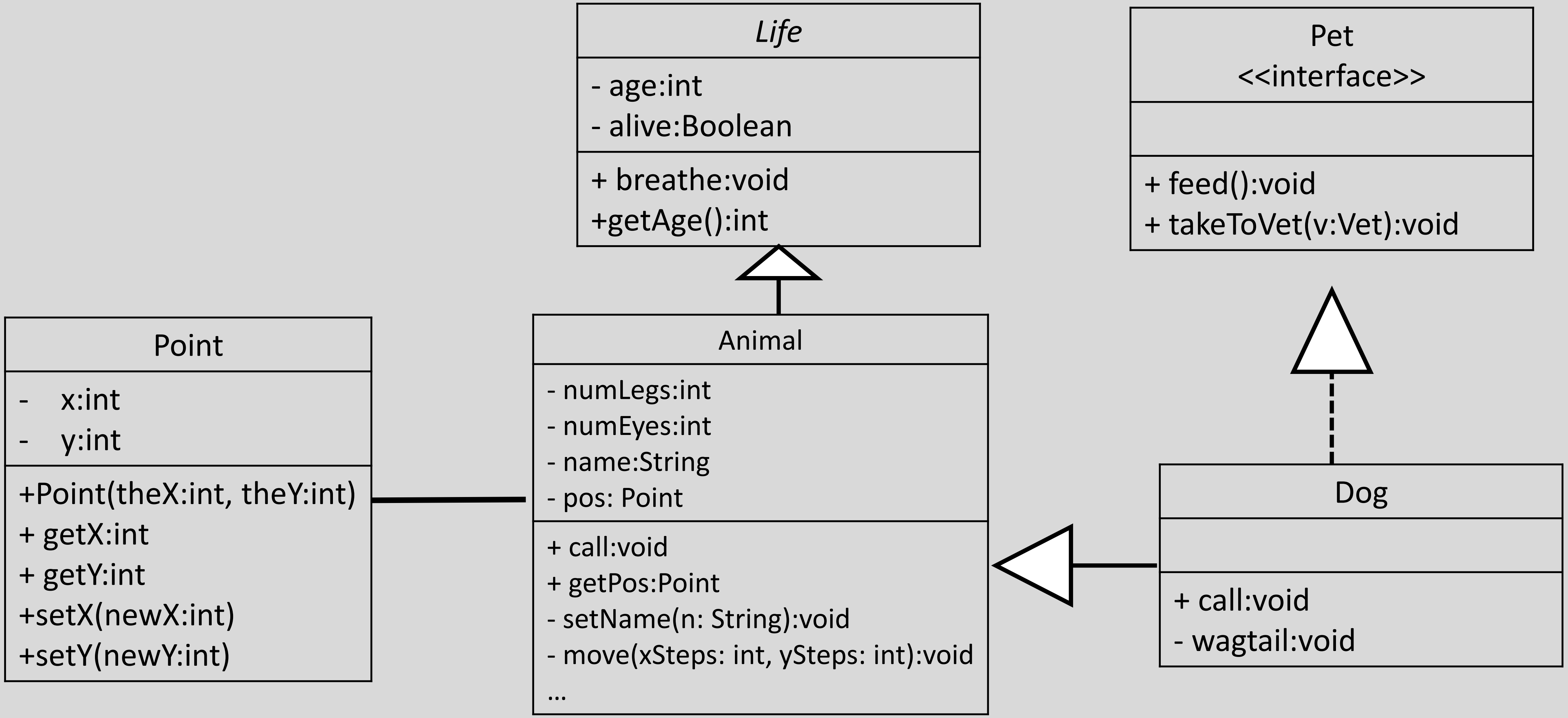
Abstract Classes

TASK: Extend Java Skeleton classes



Interfaces

TASK: Extend your Java Skeleton classes



From Pseudocode to Actual Code

TASK: Create this as a method

```
Read i
Read j
For i = 1 -> n
     For j = 1 -> m
          if i < j then print “0”
          if i == j then print “1”
          if i > j then print “2”
     End For
End For
```

From Formal Spec to Code

TASK: Create this as a method

```
Fib(0) = 0
Fib(1) = 1
Fib(n) = Fib(n-1) + Fib(n-2)
```