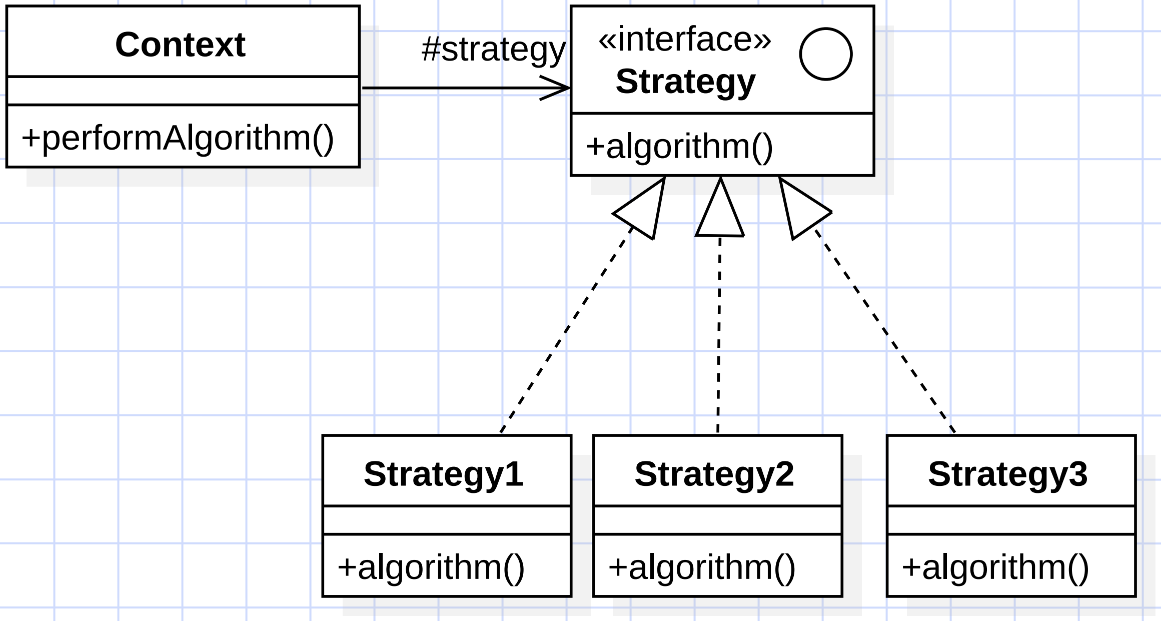
Strategy Pattern

Fromhttps://www.linkedin.com/pulse/strategy-execution-john-depalma

Strategy Pattern Model

- Definition of Strategy Pattern
  - Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.



Strategy pattern

- **strategy**: Algorithm(s) separated from a class, and **encapsulated as separate class**.
- each strategy implements **one behavior**.
- allows changing an object's behavior dynamically without extending / changing the object itself

- examples:
  - file saving/compression algorithms:
  - layout managers on GUI containers
  - AI algorithms for computer game players

How Can We Implement Strategy Pattern

- General Format in Java:
  1) Find out which functionalities are subject to change in future
  2) Design a strategy interface
  3) Have as many as needed strategy classes to implement the interface design in (2). All functionalities that considered in (1) should appear as class that implements interface in (2)
  4) Make sure your core classes having a reference of type strategy interface.
  5) Make sure your core classes have methods to set the strategy, and all strategy method from strategy classes, as needed.

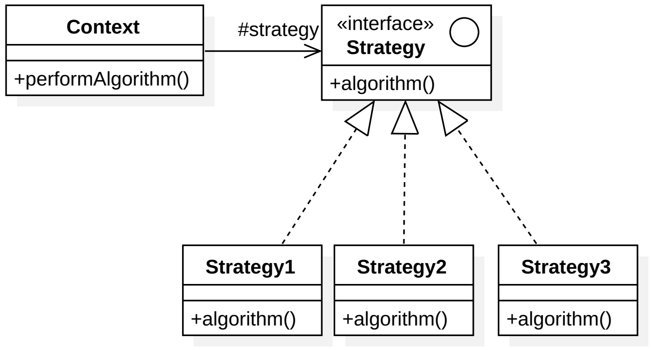
How these steps can be implemented in Java and C++?

A General and Simple Template for Strategy Pattern in Java

```
Interface Strategy{
   public void doSomething();
}
class Strategy1 implements Strategy{
 public void doSomething() {
       // implementation
       // implementation second algorithm


   }
 }
class Strategy2 implements Strategy{
public void doSomething() {
    // implementation second algorithm
   }
 }
 // MORE STRATEGIES CAN GO HERE
```

```
class Context {
   Strategy str;
  public Context() {
       // str can be set to Strategy1 by default
        // code to initialize data members
  }
  public void setStrategy1(Strategy s) {
    // s can be Strategy1 or Strategy2
     str = s;
  }
  public void performStrategy1() {
     str.doSomething();
  }
}
```



How different will be the template for C++? This question will be answered during the lecture.

Learning By Example

Developing a Hypothetical App

- Assume you are working as a software engineer with a group of developers, and you are in the process of developing a game for children that involves several core classes such as: Animal, Cat, Dog, Fish, etc. (see next slide)
- Now you need to add functionalities such as walk, swim, etc.
- We know that change of requirements during the lifetime of the software is inevitable, and as a good software designer, your code must be well-designed for possible future changes.

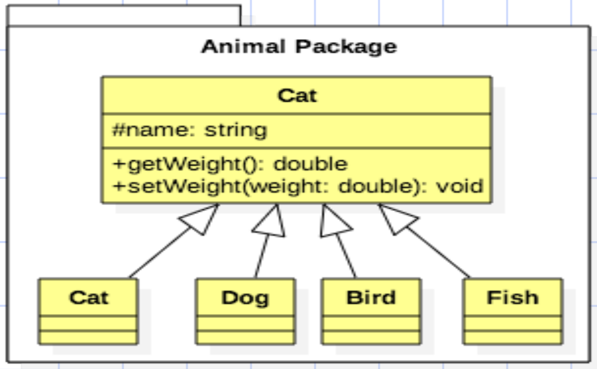
  Example:

  What if, in future you want to allow objects of class Cat not only to be able to walk, but also to be able to swim. Obviously, in a game for children and in a virtual world, this is not an unusual requirement.

Here are the core classes

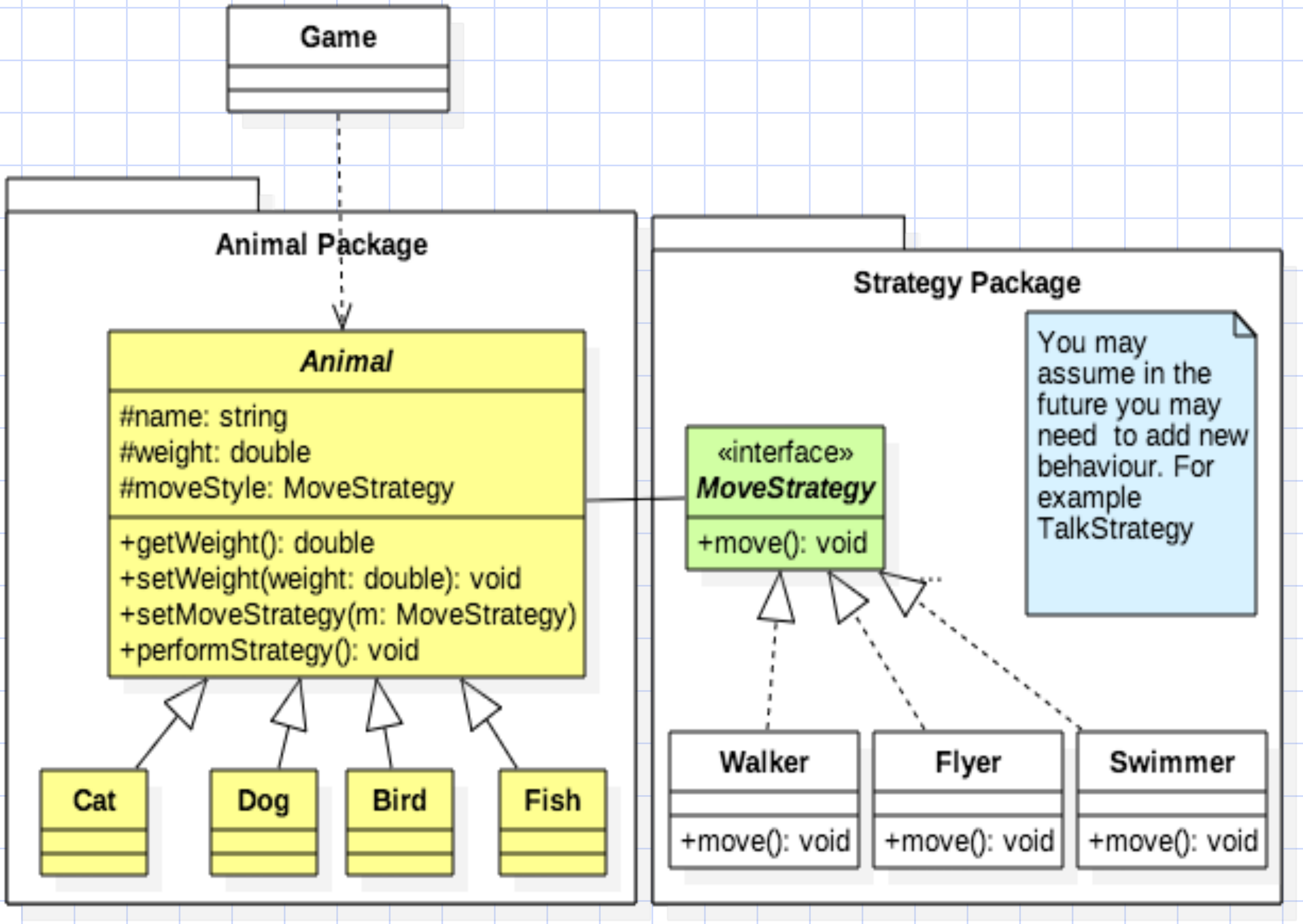
```
class Animal {
        public Animal(double wi){
                weight = wi;
                name = "";
        }
        public double getWeight() {
                return weight;
        }
        public void setWeight(double weight) {
                this.weight = weight;
        }
        protected double weight;
        protected String name;
};
```

```
class Cat extends Animal {
        public Cat(double w) {
                super(w);
                 name = "Cat";
        }
}
//-----------------------------------------------------------
class Dog extends Animal {
        public Dog(double w) {
                super(w);
                name = "Dog";
        }
}
//-----------------------------------------------------------
class Bird extends Animal {
        public Bird(double w) {
                super(w);
                name = "Bird";
        }
}
//-----------------------------------------------------------
class Fish extends Animal {
        public Fish(double w) {
                super(w);
                name = "Fish";
        }
}
```



Now, let's apply strategy pattern for animal movements.

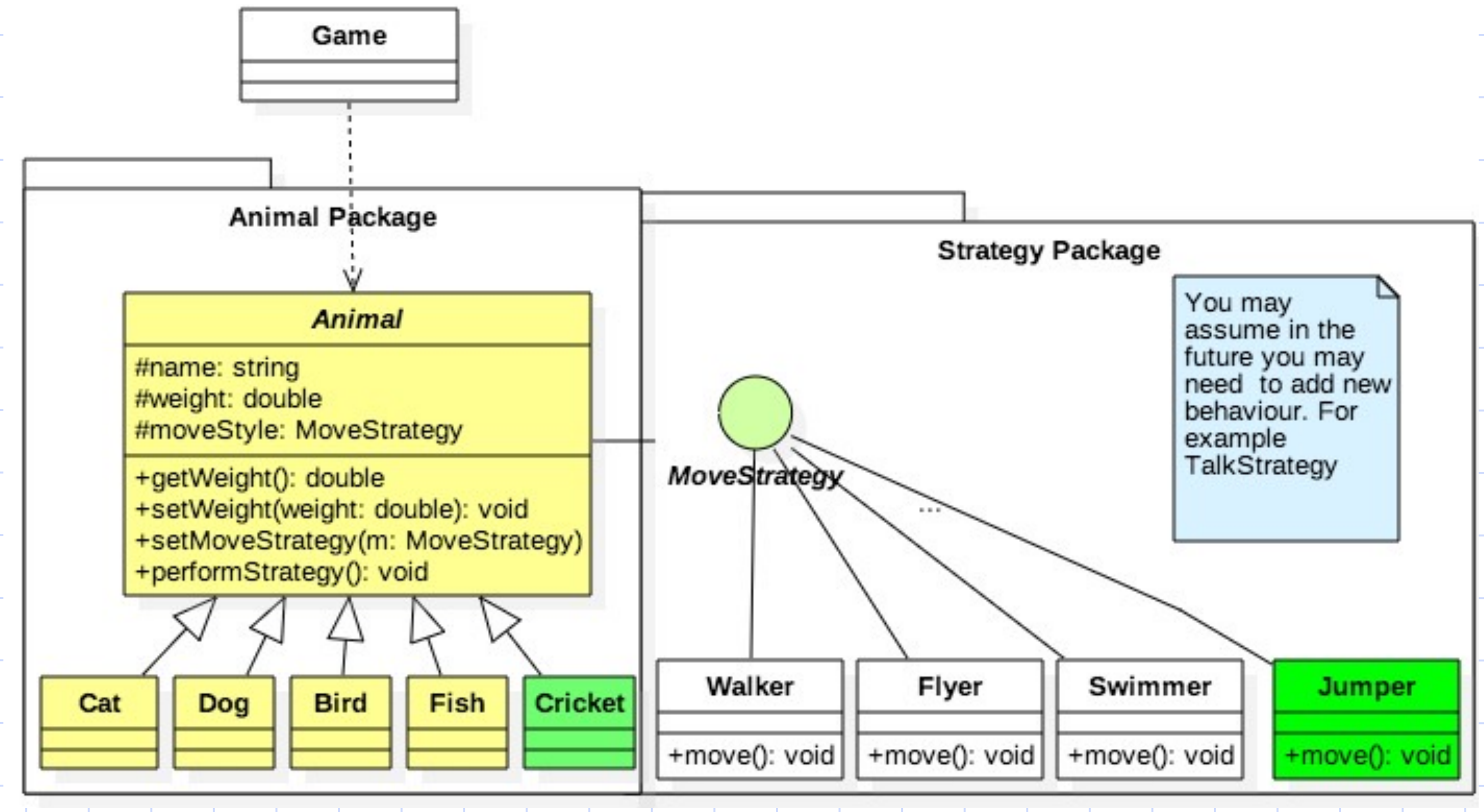
A Possible Design that Uses Strategy Pattern



From this diagram the Strategy Packsge can be easily implemented in Java or C++

Easy to Add a New Subclass

- A new subclass such as Cricket can be added without any changes to the the existing core classes.



Implementation of this application and further details will be discussed during lectures.

Summary of the Lessons Learned

- To be able to change the objects behavior without any changes to the core code of our game (Animal Hierarchy):
  - Separate changeable behaviors
  - Program to interface not implementation
  - Create concrete classes responsible for changeable behaviors.

Benefits and Drawbacks of Strategy Pattern

- Benefits:
  - A hierarchy of Strategy classes creates a family of algorithms that are reusable.
  - It is better than sub-classing, as changeable behaviors are not hardwired into context.
  - Code can be cleaner and sometime reduces that complexity of selecting a desired behavior at the runtime.
  - Can provide different implementation of the same behavior.
- Some drawbacks:
  - Communication overhead.
  - Increased number of classes, and consequently objects in the application.