Object Oriented Design and Analysis
CPE 372

Lecture 12

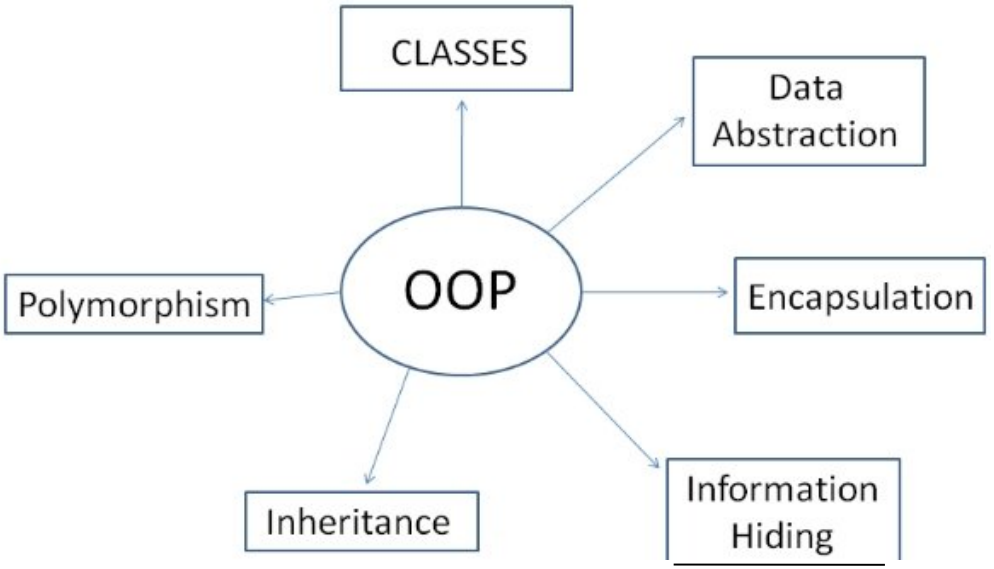
Other Object Oriented Languages

Dr. Sally E. Goldin
Department of Computer Engineering
King Mongkut's University of Technology Thonburi
Bangkok, Thailand

OOD and Java strongly connected

UML and OOD methods became popular around the same time Java was introduced

UML tools that generate code almost always generate Java



≠

Many programming languages are "object-oriented" or have some object-oriented features

JavaScript

You ***can*** implement an OO design in C

Remember Abstract Data Types (ADT)?
This is linkedListUtil.h, a linked list "class"

```
/* opaque pointer to a list */
typedef void* LIST_HANDLE;

/* Creates a new list, empty list. */
LIST_HANDLE newList();

/* Frees all memory associated with this list */
void listDestroy(LIST_HANDLE list);

/* Find out how many items currently are stored in the passed list */
int listSize(LIST_HANDLE list);

/* Add a new element to the end of a list  */
int listInsertEnd(LIST_HANDLE list, void * data);

/* Removes the element at a specified position and returns its data. */
void* listRemove(LIST_HANDLE list, int position);

/* Resets the "current" list pointer to the beginning of the list. */
int listReset(LIST_HANDLE list);

/* Returns the data stored at the "current" list position, then moves
 * the current position to the next position in the list. */
void* listGetNext(LIST_HANDLE list);

/* Find out if the current list position is past the end. */
int listAtEnd(LIST_HANDLE list);
```

C does not support key OO features

No information hiding

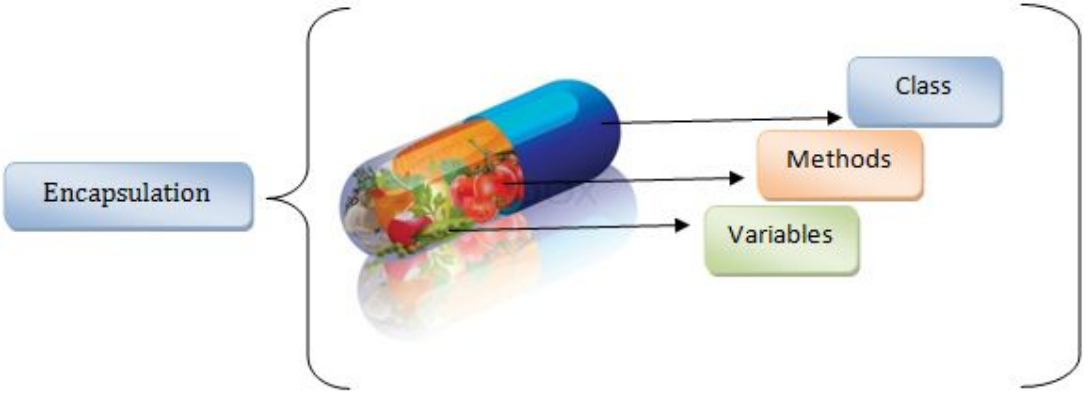
Other modules can call “private” functions

No overloading

A function can have only one set of arguments

No inheritance

But one module can reuse capabilities in another module



Encapsulation in C depends on programmer discipline

Dimensions of Comparison

"Pure" object-oriented? Or are classes and objects optional?

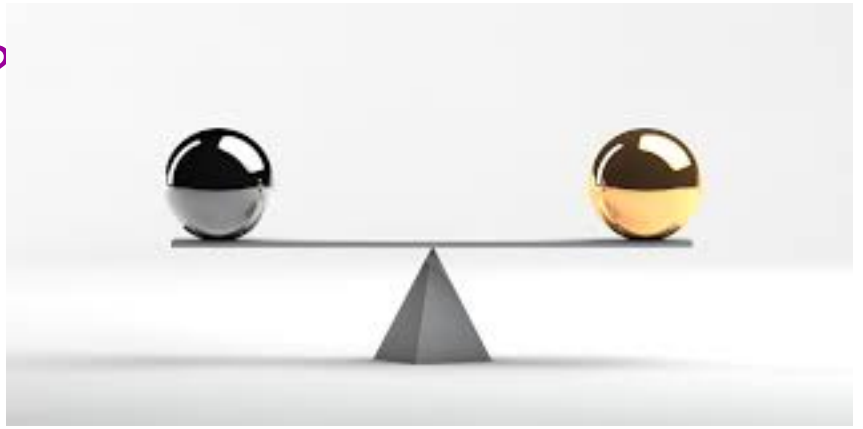
Compiled or interpreted? (Or using a virtual machine like Java?)

Object-oriented features supported:

- Classes and instances?
- Information hiding?
- Inheritance – single? multiple?
- Polymorphism and overloading?

Other features

- Strong typing?
- Garbage collection?
- Explicit exception handling?



Smalltalk

First widely disseminated object-oriented language

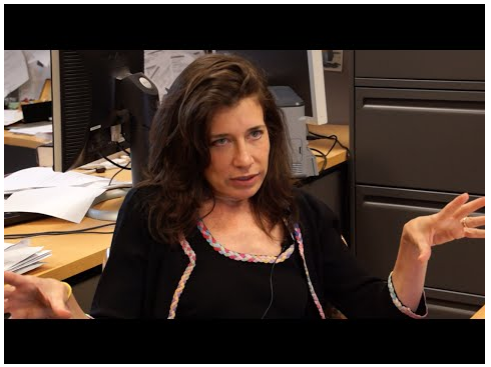
Developed at Xerox Parc research lab in the 1970's

Became publicly available (and standardized) in 1980

Characteristics:

- Pure OO – ***everything*** is a class
- ***All*** data is private
- Behavior occurs by class instances sending messages
- Interactive, interpreted - graphical UI for rapid prototyping
- Pioneered *reflection*, *garbage collection*

Adele Goldberg and Alan Kay, creators of Smalltalk



Reflection – a side note

Reflection is the ability of objects in an OO language to discover and report on their own classes and methods.

Every Java object has method *getClass()* which returns an object of class ***Class***.

Class has reflective methods like:

Method[ ] getMethods()

Constructor<?> getConstructors()

Method getMethod(String methodname,Class<?> params)

Method has a method *invoke() -* so you can call it!

Reflection in Exercise

We saw a simple example of reflection in Exercise 4.

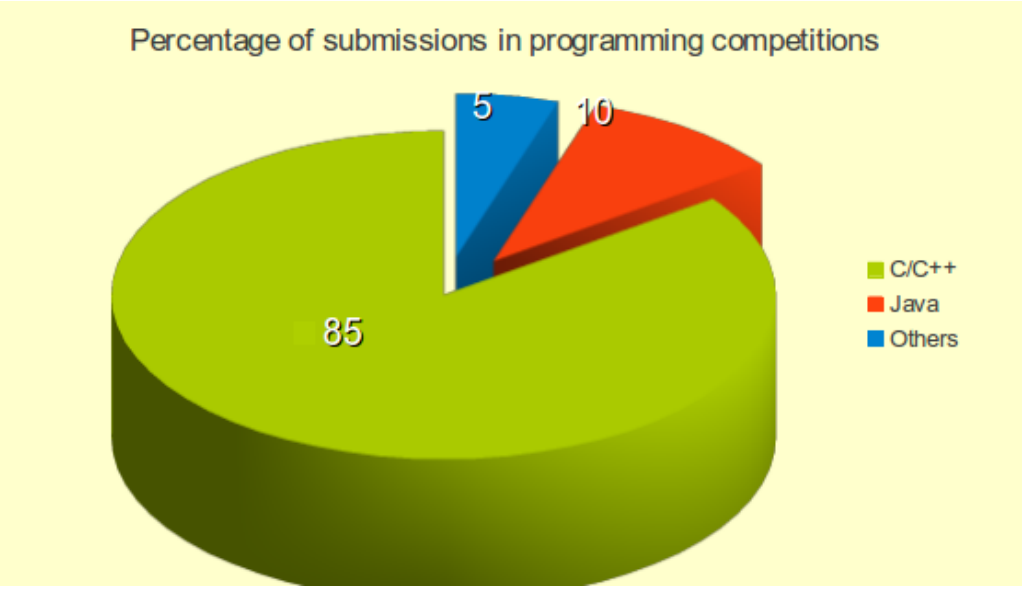
From ***ShapeFileTester.java***

```
AbstractShape nextShape = reader.readShape();
while (nextShape != null)
{
    System.out.println("  readShape returned an object: "
        + nextShape.getClass().toString());
    System.out.println("      toString: "
        + nextShape.toString());
    System.out.println("      perimeter: "
        + nextShape.calcPerimeter());
    nextShape = reader.readShape();
}
reader.close();
```

nextShape **is an *AbstractShape*, a superclass. We use getClass().getString() to print out the specific subclass (*Square, Diamond,* etc.)**

C++

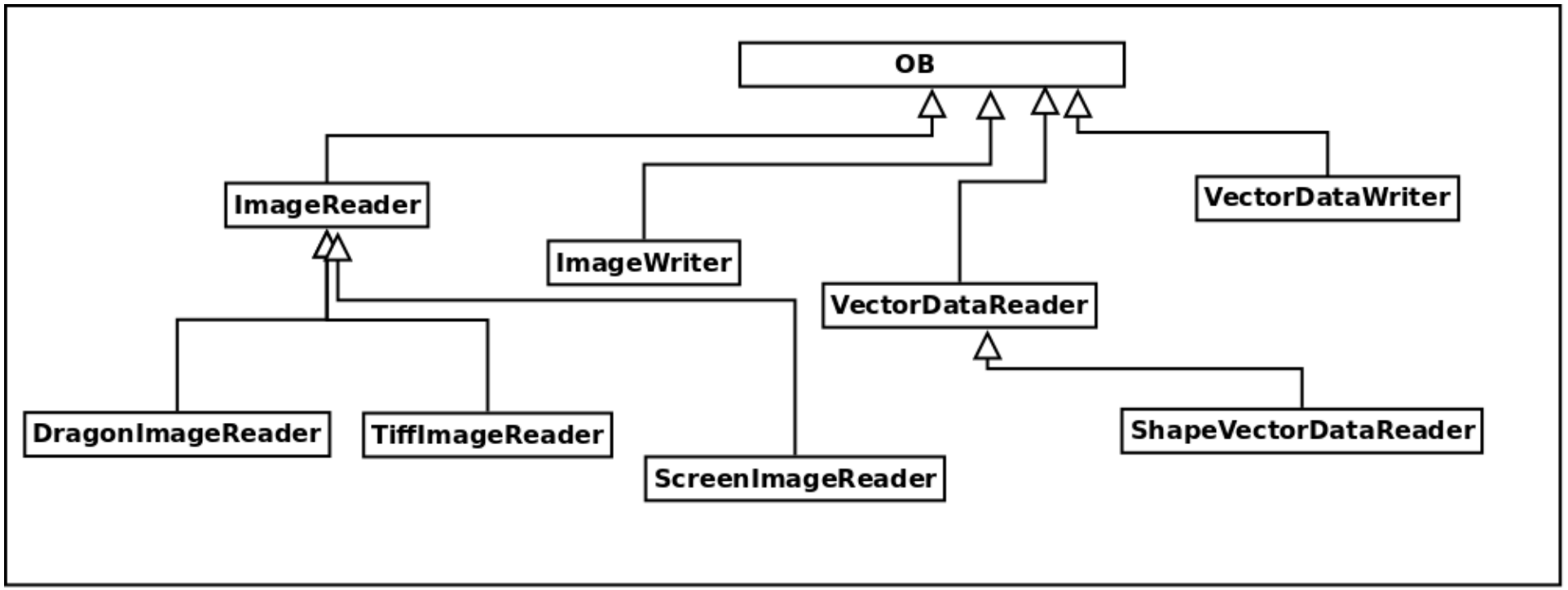
- Created in 1979 by Bjarne Stroustrup
- Extends standard C by adding object-oriented features
- Emphasis on fast execution, ability to directly access hardware
- As of 2017, C++ remains the third most popular programming language, behind Java and C.



C++ Characteristics

- Not “pure” - use of OO features is optional
- Compiled – not necessarily portable
- Strongly typed
- Multiple levels of visibility (different in detail from Java)
- Allows multiple inheritance
- Provides **`try...catch`** exception handling (like Java)
- No garbage collection; provides explicit “destructors”
- Separates the class definition (in the header file) from the implementation (in the .cpp file) but can have “inline” functions in the header file

Example: Partial Dragon Class Hierarchy



*See code in **demos/Lecture12/dragonCpp** and **demos/Lecture12/dragonCpp/include***

JavaScript

Originally a scripting language to provide interactivity for web applications

Created at Netscape Communications, standardized as ECMAScript in 1997 (“scripting language wars”)

Increasingly used as a server side language

“Multi-paradigm”: event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles.

JavaScript Characteristics

- Interpreted – no compile step (weak error checking)
- Loosely typed – variables must be declared but data types are implicit and can change
- Objects created dynamically – member data items (“properties”) not fixed but can be added at any time (no concept of class)
- Inheritance supported via ***prototypes*** **–** however, the prototype is just a property referencing another object

JavaScript Prototype Example 1

```
let animal = {
  eats: true
};
let rabbit = {
  jumps: true
};

rabbit.__proto__ = animal; // (*)

// we can find both properties in rabbit now:
alert( rabbit.eats ); // true (**)
alert( rabbit.jumps ); // true
```

In this example, `animal` and `rabbit` are both objects. The `animal` object has the property `eats` with a value of true.

When we set the special `__proto__` property of `rabbit` to `animal`, `rabbit` inherits the `eats` property (and its value).

from *https://javascript.info/prototype-inheritance*

JavaScript Prototype Example 2

```
let animal = {
  eats: true,
  walk() {
    alert("Animal walk");
  }
};

let rabbit = {
  jumps: true,
  __proto__: animal
};

// walk is taken from the prototype
rabbit.walk(); // Animal walk
```

As shown in this example, `animal` can also define methods which `rabbit` can then use, because `animal` is its prototype.

from *https://javascript.info/prototype-inheritance*

JavaScript Prototype Example 3

```
let animal = {
  eats: true,
  walk() {
    alert("Animal walk");
  }
};

let rabbit = {
  jumps: true,
  __proto__: animal
};

let longEar = {
  earLength: 10,
  __proto__: rabbit
}

// walk is taken from the prototype chain
longEar.walk(); // Animal walk
alert(longEar.jumps); // true (from rabbit)
```

This example illustrates a multi-level inheritance structure. The object `longEar` is a rabbit which is an `animal`. Thus it can call functions ("methods") or access properties ("members") of both `rabbit` and `animal`.

from *https://javascript.info/prototype-inheritance*

JavaScript Prototype Example 4

```
let animal = {
  eats: true,
  walk() {
    /* this method won't be used by rabbit */
  }
};

let rabbit = {
  __proto__: animal
}

rabbit.walk = function() {
  alert("Rabbit! Bounce-bounce!");
};

rabbit.walk(); // Rabbit! Bounce-bounce!
```

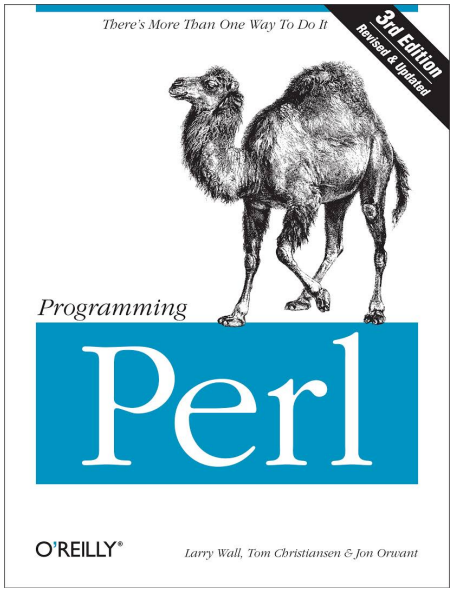
Values of inherited properties are read-only. So I can set `rabbit.eats` to be false, without affecting the value in the `animal` prototype (or any other objects that use it).

Meanwhile, as shown above, if I define a function with the same name as an inherited function, JavaScript will call the local, not the inherited version.

This is like overriding a method. JavaScript does not directly support overloading. However, it also doesn't check the number or type of function arguments. So you can write one function with branches for different passed arguments.

Perl

- Originally developed in 1987 by Larry Wall for Unix scripting and reporting
- Borrowed from many other languages including C, shell scripting, awk and sed
- The “camel book” came out in 1991 (first comprehensive doc)
- Currently split into Perl 5 and Perl 6, which are actually two different (though related) languages developed by different (open source) projects
- “Swiss Army chainsaw of scripting languages”
- “Duct tape that holds the Internet together”
- Widely used for:
  - Web app development (CGI, server side logic)
  - System administration
  - Text processing and analysis



Perl Characteristics

- Interpreted (but somewhat better error checking than JS)
- Loosely typed, clear scope rules
- “Lazy” syntax – many styles (“write-only language”)
- Powerful string manipulation capabilities using regular expressions
- Hashes (associative arrays) a fundamental data type
- Extensible using “Perl Modules”
- Huge repository in CPAN (Comprehensive Perl Archive Network)

CPAN

Object Oriented Features in Perl

Only through the use of modules

```
# Tell Perl the modules to import
use CGI::Carp qw(fatalsToBrowser);
use CGI qw(param);
use DBI;
```

Create a CGI instance to use for parsing arguments

```
my $q = CGI->new;
```

Call methods on the instance, e.g. to get arguments passed from an HTML form

```
my $gCourse = $q->param('course');
my $gButton =  $q->param('button');
my $gStudentId = $q->param('studentid');
my $gSection = $q->param('section');
my $gLab = $q->param('lab');
```

See the full Perl program at demos/Lecture12/attendanceTool.pl

Creating your own modules

Can be written in either Perl or C

If in Perl, must begin with `package` statement

```
package TLKS;
```

Constructor should be called `new`

```
sub new
    {
    my $self = {};
    $$self{dbDbi} = new GRS::DB_DBI_Access;
    $$self{errStr} = '';
    return bless $self;
    }
```

Create other methods as desired, using `sub`

Must end with 1 (true)

See demos/Lecture12/tlks.pm

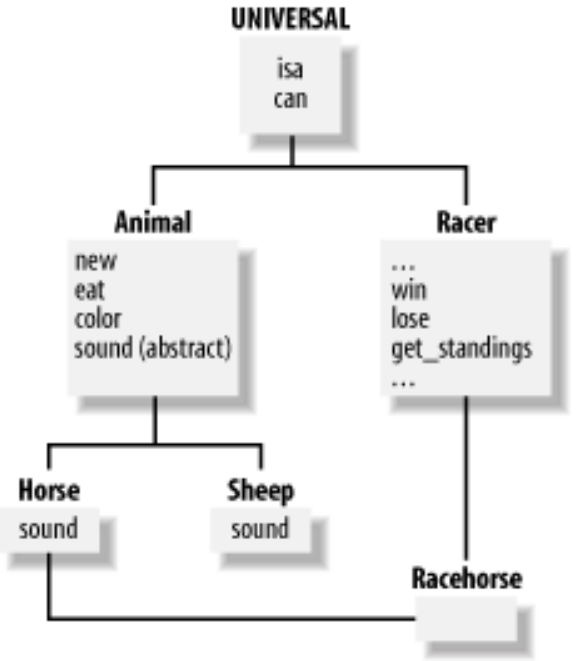
Perl OO Capabilities

Working with objects and classes is optional in Perl

- Not “pure” OO
- Alternative non-OO syntax for using modules

However, Perl provides ways to implement all the familiar characteristics of object oriented programming

- Classes, instances and constructors
- Encapsulation
- Inheritance (including multiple inheritance!)
- Polymorphism
- Overloading
- etc.



From: *http://etutorials.org/Programming/Perl*

The Perl Philosophy

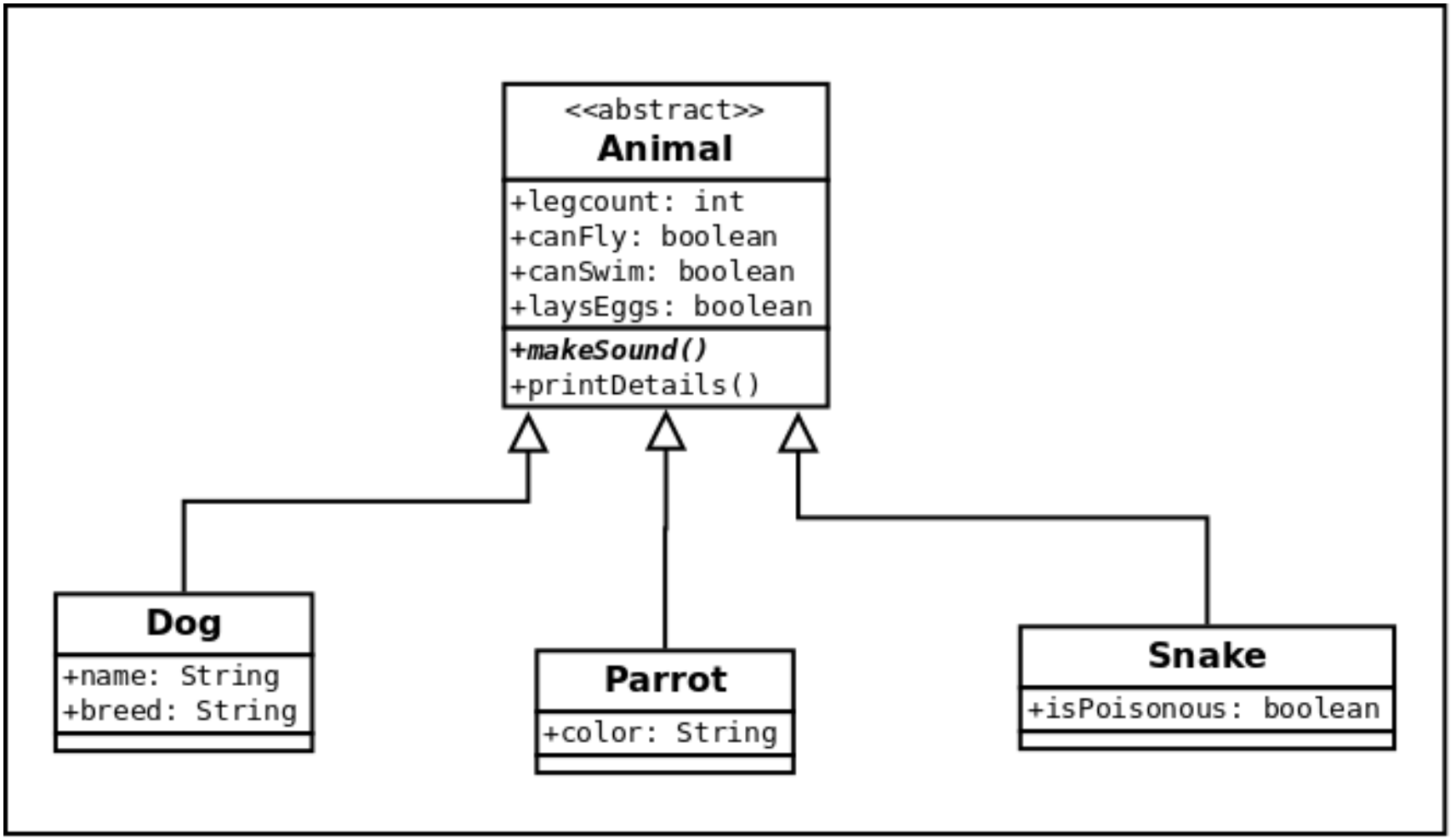
“Perl doesn't force a particular style of programming on you, and it doesn't have the obsession with privacy that some other object-oriented languages do. Perl does have an obsession with freedom, however, and one of the freedoms you have as a Perl programmer is the right to select as much or as little privacy as you like. In fact, Perl can have stronger privacy in its classes and objects than C++. That is, Perl does not restrict you from anything, and in particular it doesn't restrict you from restricting yourself, if you're into that kind of thing.”

Learning Perl, Third Edition
by Larry Wall, Tom Christiansen, and Jon Orwant

See: *https://docstore.mik.ua/orelly/perl/prog3/ch12_01.htm*

Exercise

Choose your favorite OO programming language and implement the (silly) class hierarchy below.



Exercise continued

Notes:

I have shown the member data items as public, but if you can create them as private and implement getters and setters, that is preferred

You must also implement a test driver that will create instances of the ***Dog***, ***Parrot*** and ***Snake*** classes, set their data appropriately, and call the makeSound and printDetails methods.

For the makeSound method, just print the sound the animal makes: “bow wow” (or “hong hong” if you prefer the Thai!), “squawk”, and “hiss”.