

INTRODUCTION TO VISUAL AND INTERACTIVE PROGRAMMING

CT803-4-0-OIVIP

Topic: Object-Oriented Programming

Topic Learning Outcomes

At the end of this topic, you should be able to:

- Describe the basics of object-oriented programming.
- Implement clone, constructor and method in block programming.

Contents & Structure

- Object oriented
- Defining Classes and Objects
- Object-Oriented Programming Concepts and Terminology
- Object oriented in Block Programming
- Working with Object

Object Oriented Programming - OOP

“Object-oriented programming is an approach that provides a way of molding programs by creating partitioned memory area for both data, and functions that can be used as templates for creating copies of such modules on demand.”

Object-oriented programming - General



Also known as **OOP**, is a programming paradigm.



It revolves around data structures called *objects*, which consist of states and behaviors, and the interaction between them via *message passing*; messages and variables are the two main kinds of abstraction used in OOP.

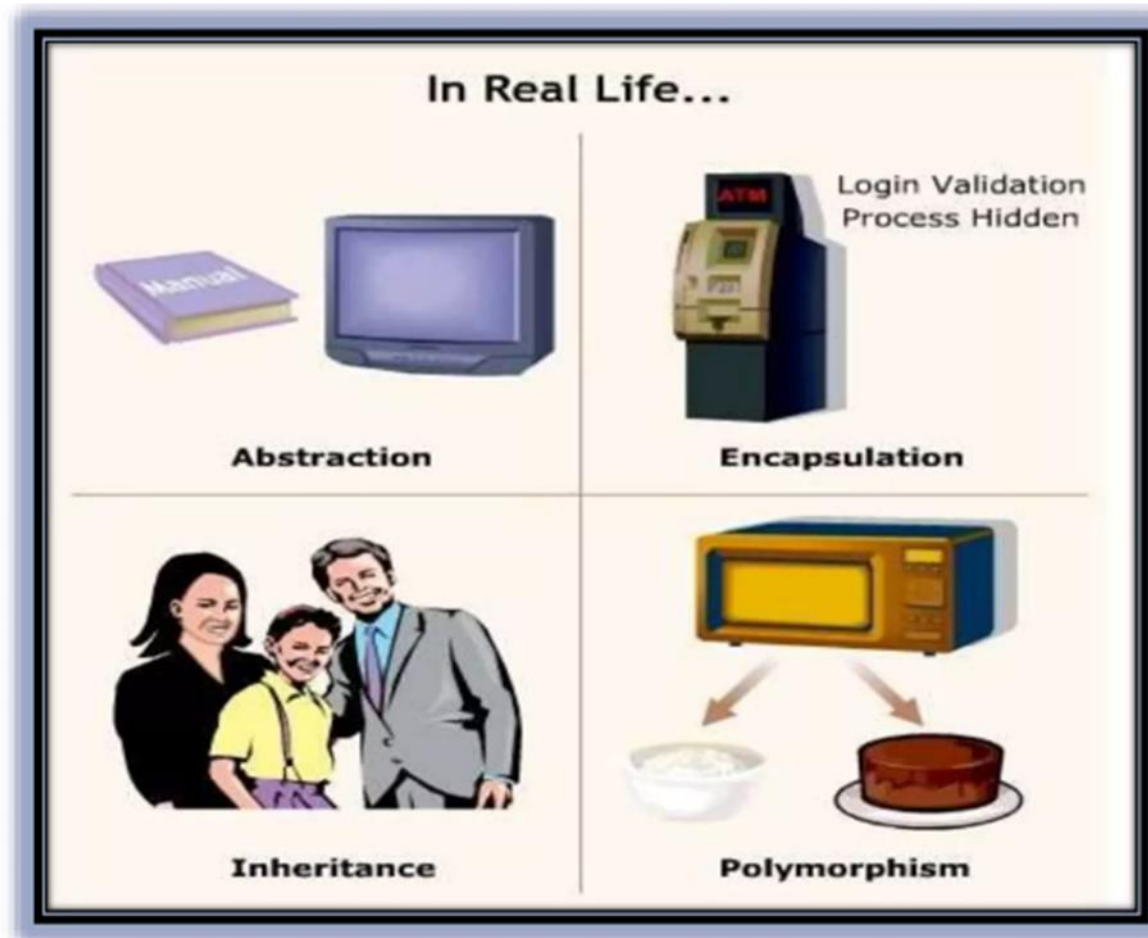


Rather than write a set of procedures to manipulate members of each data type, the behavior becomes part of the data itself.



This makes many kinds of program, particularly ones involving simulation or graphical interfaces, easier to write.

Basic Concept



Defining Classes and Objects

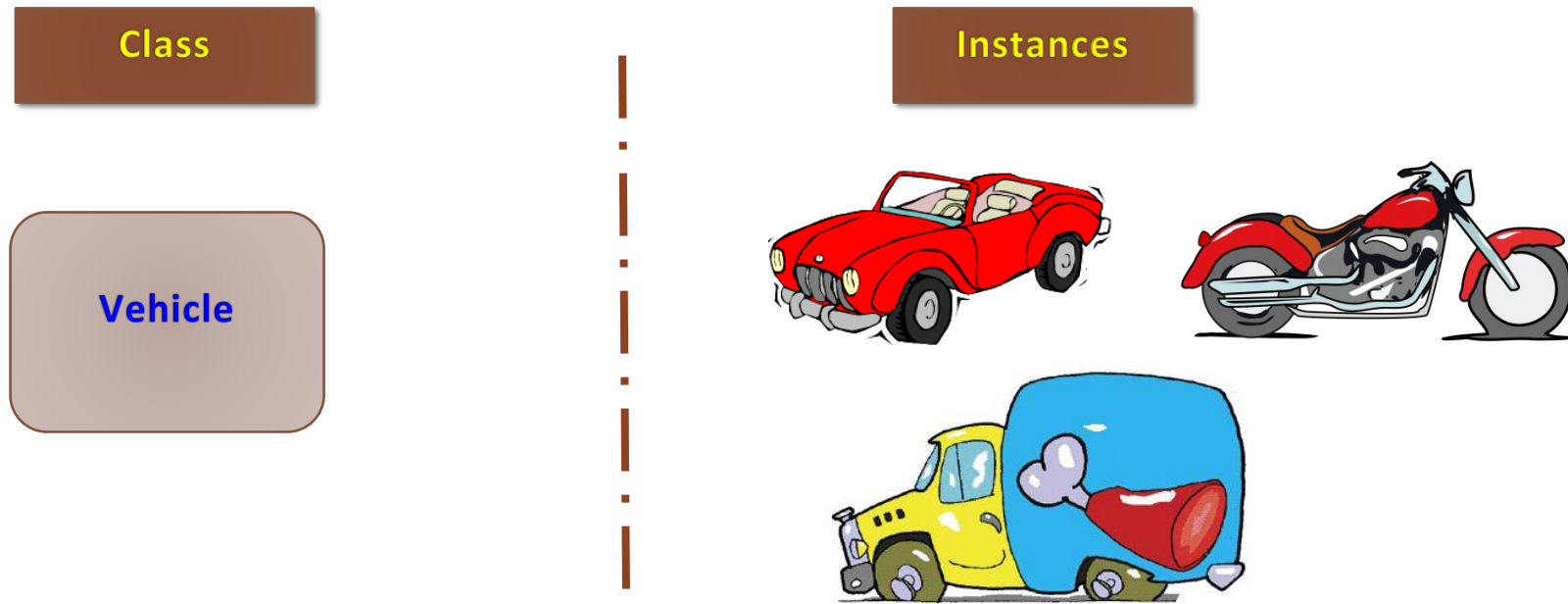
Classes and Objects are basic concepts of Object-Oriented Programming which revolve around the real-life entities.

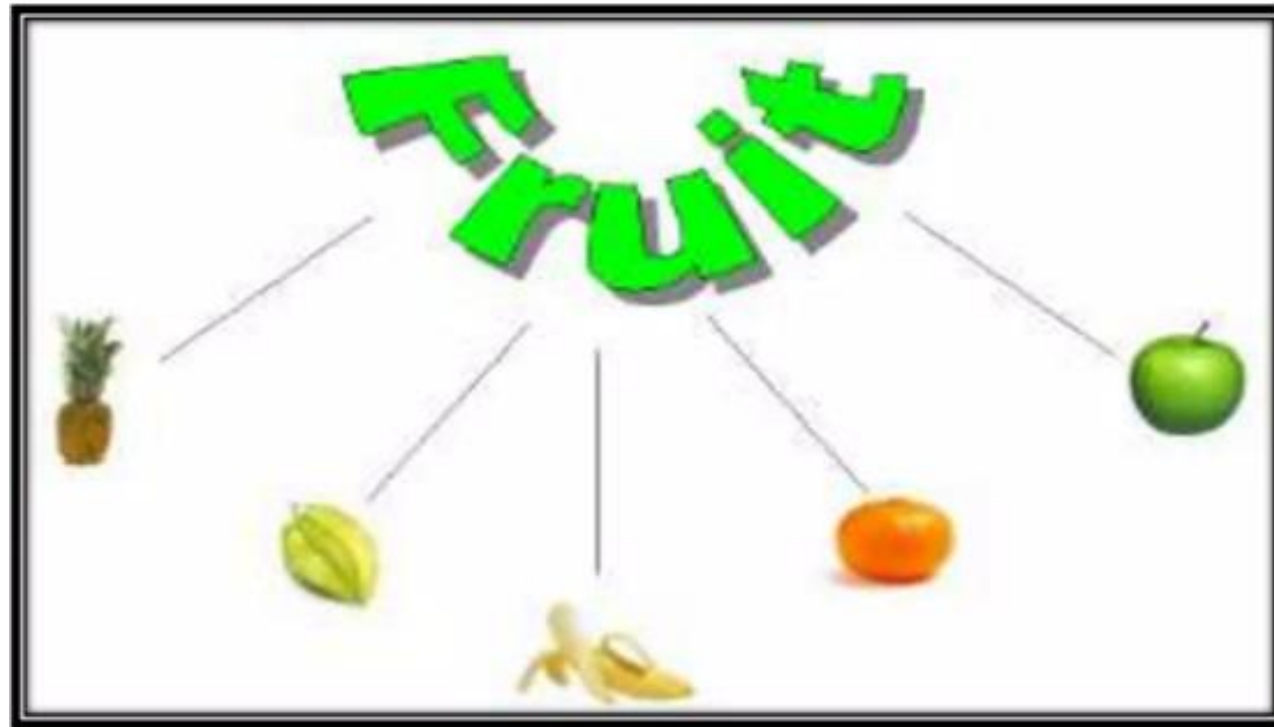
Class and Object

- Class
 - A class is a group of objects which have common properties.
 - It is a template or blueprint from which objects are created.
 - It is a logical entity.
 - It is non primitive data type. –
 - It cannot be physical(no memory space)
 - Class members are access modifiers, objects , Methods , Instance variable and constructors.
- Object
 - An Object is an Instance of a class
 - Any entity that has state and behavior is known as an object.
 - For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical

Defining Classes and Objects

- Class is a collection of **similar** objects.
- The object that satisfies the class definition of a particular class are said to be an instantiate of the class.





Object



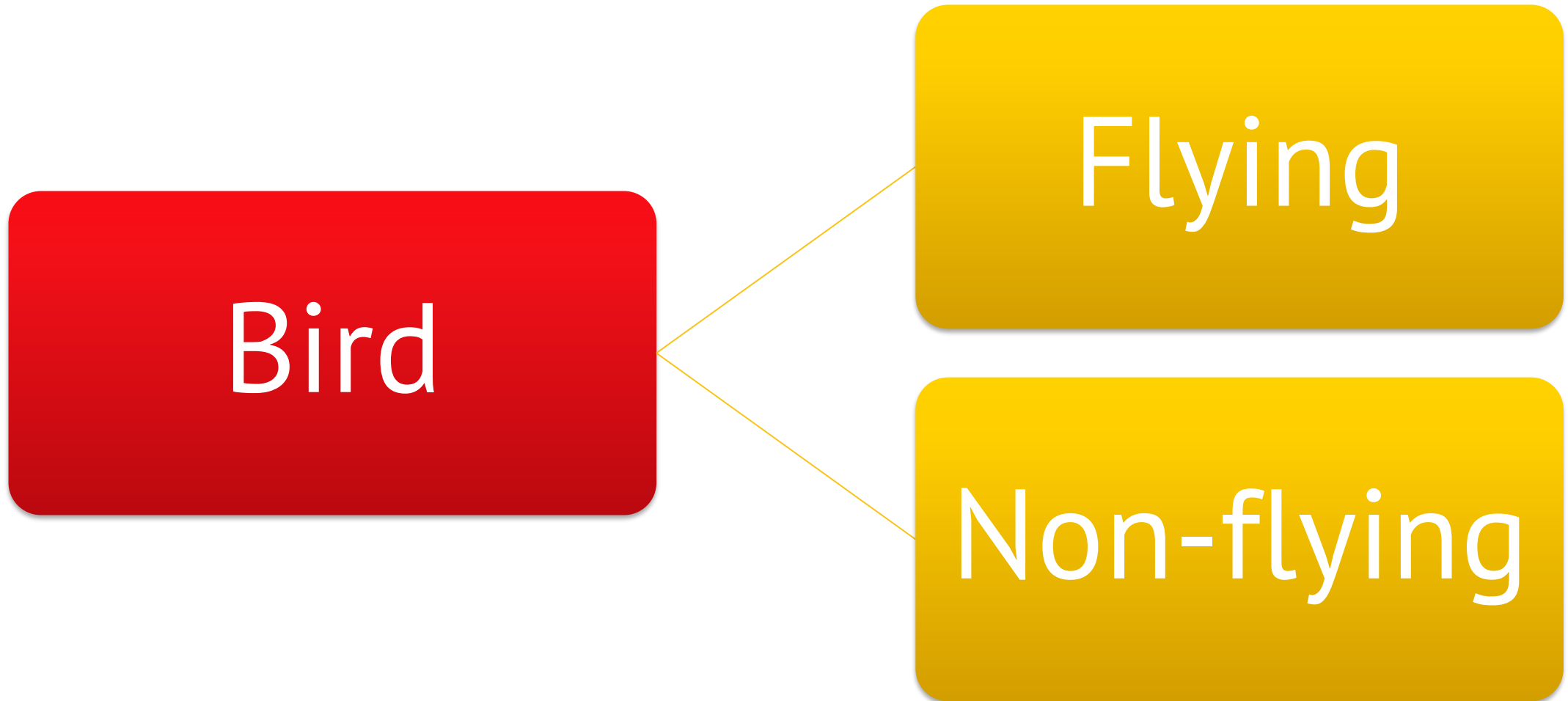
Dog Properties

Color
Eye-color
Height
Length
Weight

Dog Behavior

Bark
Sit
Lay Down
Come
Shake

Inheritance



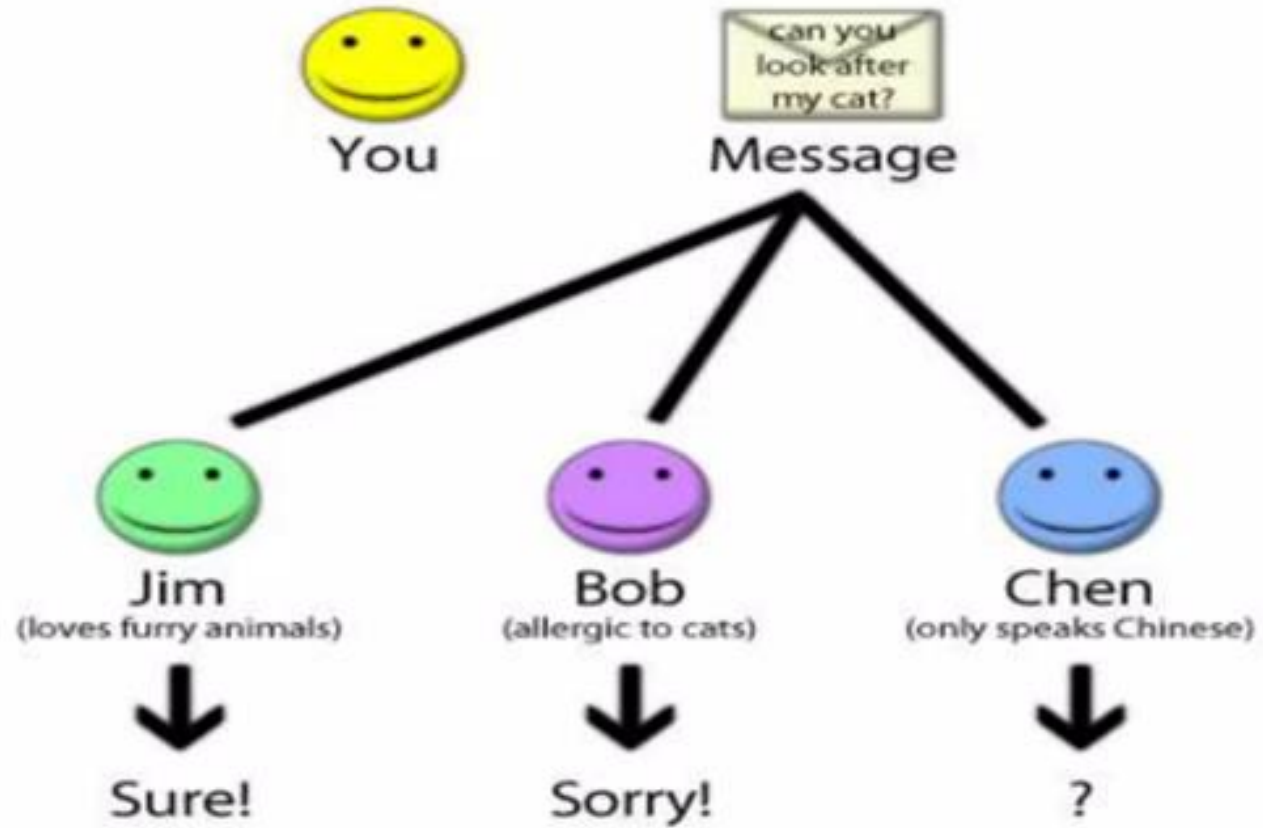
Data Abstraction and Encapsulation



Encapsulation:
Calculator shows the result of equation but hides the implementation(Calculating the result) involved

Abstraction:
The calculator shown in the figure must be powered by a battery source. How the battery works for the calculator is not necessary to know.

Message Passing



Class-based OOP

The traditional form of object-oriented programming is called class-based OOP.

A class represents a type;

Everything is an object

Objects communicate via messages (handled by methods)

Objects have their own state

Every object is an instance of a class

A class describes its instances' behavior

Prototype-based OOP



Classes are not necessary; instead, objects extend other objects, called their prototypes, and variables and methods may be added to any object at any time.



If an object does not understand a message, it is automatically delegated to its prototype



There are two methods of constructing new objects: object creation "from nothing" or through cloning an existing object.



Example – BYOB and SNAP!

Object Oriented with Sprites

- A style based around the abstraction object: a collection of data and methods (procedures, which from our point of view are just more data) that you interact with by sending it a message (just a name, maybe in the form of a text string, and perhaps additional inputs).
- Object responds to the message by carrying out a method, which may or may not report a value back to the asker.
- Snap! approach is less restrictive than that of some other OOP languages; allowed objects easy access to each others' data and methods.


Object Oriented with Sprites

- Technically, object-oriented programming for BYOB and SNAP! Is based on these three
 - **Message passing:** There is a notation by which any object can send a message to another object.
 - **Local state:** Each object can remember the important past history of the computation it has performed. (“Important” means that it need not remember every message it has handled, but only the lasting effects of those messages that will affect later computation.)
 - **Inheritance:** It would be impractical if each individual object had to contain methods, many of them identical to those of other objects, for all the messages it can accept.

First Class Sprites

- Snap! sprites are first class data.
- They can be created and deleted by a script, stored in a variable or list, and sent messages individually.
- The children of a sprite can inherit sprite-local variables, methods (sprite-local procedures), and other attributes (e.g., x position).

Permanent and Temporary Clones

- The  block is used to create and report an instance (a clone) of any sprite. (There is also a command version, for historical reasons.) There are two different kinds of situations in which clones are used.
 - Made an example sprite and, when project start, you want a fairly large number of essentially identical sprites that behave like the example. (The example sprite called as the “parent” and the others the “children.”) These are **temporary** clones. They are automatically deleted when the user presses either the green flag or the red stop sign.

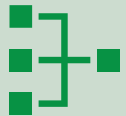
Permanent and Temporary Clone

- Need or want specializations of sprites.
 - For example, you have a sprite named Dog. It has certain behaviors, such as running up to a person who comes near it.
 - You decide that the family in your story really likes dogs, so they adopt a lot of them. Some are cocker spaniels, who wag their tails when they see you. Others are rottweilers, who growl at you when they see you.
 - So, you make a clone of Dog, perhaps rename it Cocker Spaniel, and give it a new costume and a script for what to do when someone gets nearby.
 - You make another clone of Dog, perhaps rename it Rottweiler, and give it a new costume, etc.
 - Then you make three clones of Cocker Spaniel (so there are four altogether) and two clones of Rottweiler.
 - Maybe you hide the Dog sprite after all this, since it's no breed.
 - Each dog has its own position, special behaviors, and so on.
 - You want to save all these dogs in the project.
 - These are **permanent** clones.

Sending Message to Sprites

- The messages that a sprite accepts are the blocks in its palettes, including both all-sprites and this-sprite-only blocks.
- The way to send a message to a sprite (or the stage) is with the **tell block** (for command messages) or the **ask block** (for reporter messages).
- Tell and ask wait until the other sprite has carried out its method before this sprite's script continues.

Constructing Objects



Constructors are used for initializing new objects.



Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

Attribute and Method

Every object can contain attributes and methods:

Attribute

- Is a variable that is associated with one object.

Method

- Is a function or a script that causes an object to perform some action.
- A method can access or change the attributes associate with this object, or it can perform some other action.
- A special kind of method called a **constructor** defines the actions required to initialize all of the attributes of an object - when an object is created, it creates variables for all of its attributes, and just like any variables these should be initialized!

List of Attributes

my neighbors ▾

neighbors
self
other sprites
clones
other clones
parts
anchor
stage
children
parent
name
costumes
sounds
dangling?
rotation x
rotation y
center x
center y

attribute ▾

draggable?
name
rotation style
synchronous?
direction
x position
y position
costume #
costumes
hidden?
layer
size
brightness
color effect
fisheye effect
ghost effect
mosaic effect
pixelate effect
whirl effect
instrument
sounds
tempo
volume
pen color
pen down?
pen shade
pen size
anchor
children
parent
parts

A

B

- Some BYOB/SNAP! operations that work on objects. In the “Sensing” category of the blocks palette, there is a reporter block at the bottom that is named “attribute” (BYOB) and “my”(Snap!).
- If you drag that out and click the drop-down menu, you’ll see this long list:
 - Those are all the “standard attributes” that are part of any sprite!
 - You can see everything from the name of the sprite, to direction and x and y position coordinates, current costume number, and more

List of Attributes

Several of these are not real attributes, but rather *lists* of things related to attributes:

- **neighbors**: a list of *nearby* sprites
- **other sprites**: a list of all sprites except myself
- **clones**: a list of my *temporary* clones
- **other clones**: a list of my *temporary* siblings
- **parts**: a list of sprites whose **anchor** attribute is this sprite
- **children**: a list of all my clones, temporary and permanent
- **costumes**: a list of the sprite's

The others are non-lists:

- **self**: this sprite
- **anchor**: the sprite of which I am a (nested) part
- **stage**: the stage, which is first-class, like a sprite
- **parent**: the sprite of which I am a clone
- **name**: my name (same as parent's name if I'm temporary)
- **dangling?**: True if I am a part and not in synchronous orbit

Don't rely on these screwy ones which are likely to change:

- **rotation x, rotation y**: same as **x position, y position**
- **center x, center y**: the x and y position of the center of my bounding box, rounded off

Visual Programming: Clear Mainline

- Every sprite should have one and only one Green Flag script
 - Snap! allows more than one Green Flag script, but it is easy to give conflicting instructions, or to count on one being executed first
- Snap! provides blocks for running scripts in parallel, and running scripts that belong to other sprites



Run

Takes a Command (script) as its input, and carries out its instructions



Launch

Identical to run except that it calls the method as a separate script, so the calling script can meanwhile do something else

Call

Takes a Reporter or Predicate block as its input, and returns a value that can be stored in a variable or used in a condition



Summary / Recap of Main Points

- Object oriented
- Defining Classes and Objects
- Object-Oriented Programming Concepts and Terminology
- Object Oriented in Block Programming
- Working with Objects