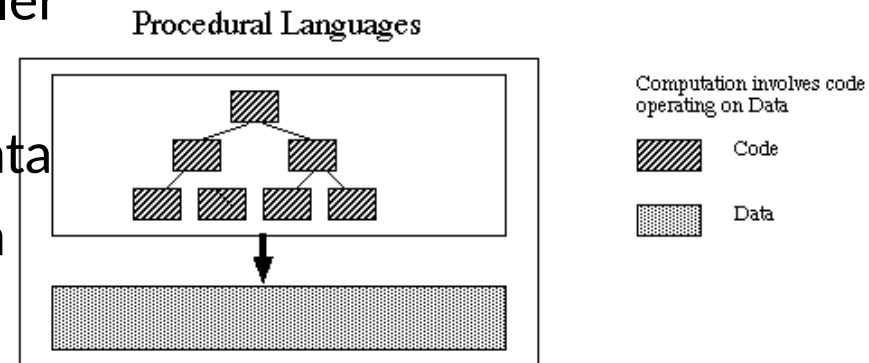


OOPs & Java

Procedural Language(POP) - ex. 'C'

- **Characteristics:**

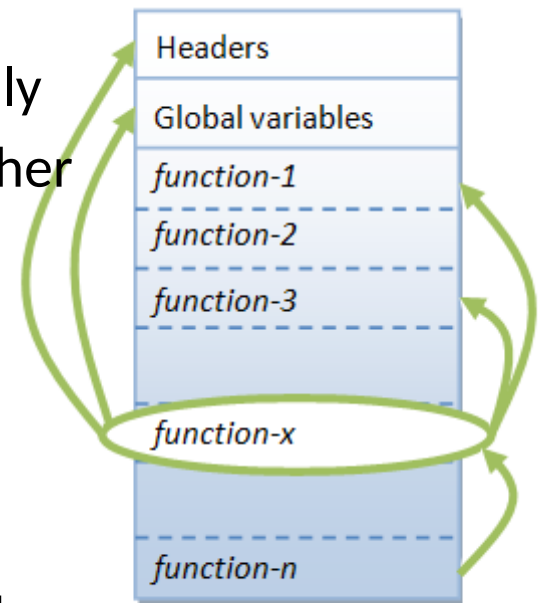
- Emphasis is on doing things (**algorithms**)
- Large programs are divided into smaller **functions**
- Most of the functions share **global** data
- Data moves openly across the system from function to function
- Functions transform data from one form to another
- Uses **top-down approach** to program design



Procedural Languages - Drawbacks

- **Drawbacks:**

- As functions openly access global data, new programmer could **easily corrupt data** accidentally
- Can easily access data from one function in another function, **no protection!**
- In large projects, it becomes difficult to identify what data is being used by which functions
- If program is modified to handle new data, all functions need to be modified to access data
- Functions transform data from one form to another
- Does not model **real-time problems** very well



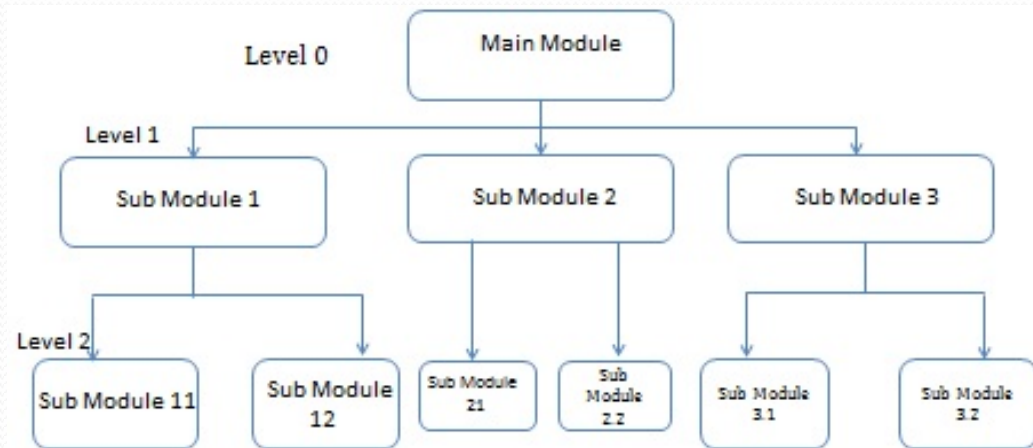
A function (in C) is not well-encapsulated

Difference between POP & OOP's

Procedural oriented programming	Object Oriented Programming
It is known as POP	It is known as OOP
It deals with algorithms	It deals with data
Programs are divided into functions	Programs are divided into objects
Most of the functions share global data	Data Structure characterizes objects
Data move from function to function	Functions that operate on data are bind to form classes
Functions are responsible for transforming from one form to another	Data is hidden cannot be accessed by outside functions
It is top down approach	It is Bottom Up approach
It needs very less memory	It needs more memory than POP
Example:- C, Fortran	Example:- C++, JAVA,.NET
It do not have any access specifiers	It has access specifiers like private, public and protected.
It is less secure	It is more secure
It follows no overloading	It follows operator overloading and function overloading

Top-down approach – ‘C’

- Overview of system is formulated
- For each part; refine (+add) details
- Keep adding details **everywhere** until system is ready
- Large program broken into smaller functions
- Complex code
- Global data focused
- Limited; to no code reuse

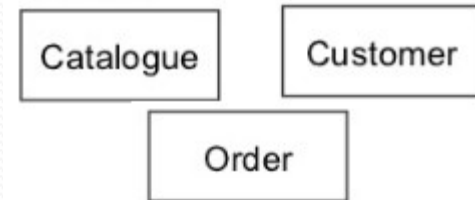


Bottom-up approach – Java

- Describe entities then how they interact
- Start with concrete business case, proceed to implementation
- Easy to develop
- Unit test it here, and validate now
- Easy to change & modify later
- No duplication of code
- Reuse, wherever needed

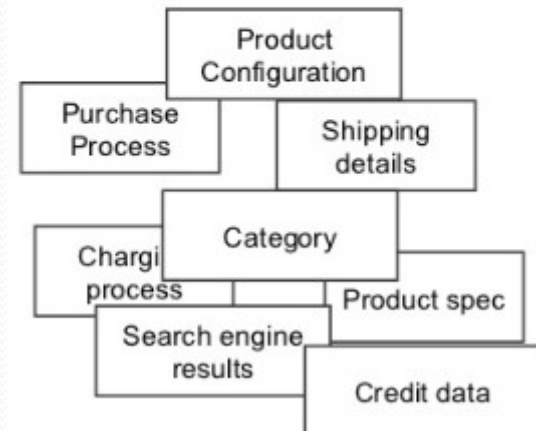
Top-down Process

Starting with an overview of the system



Bottom-up Process

Starting with throwing all classes on the page, and then combining them:



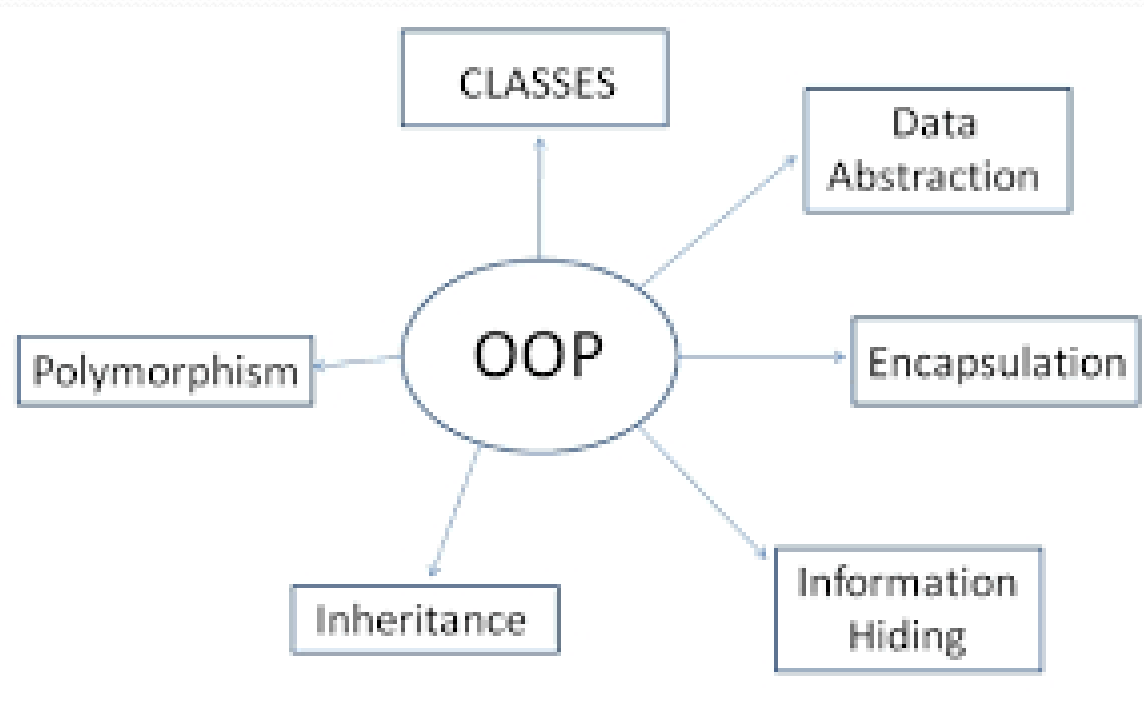
Object Oriented - Advantages

- Code reuse & recycle
- Improved s/w development productivity
- Improved software maintainability
- Lesser development time
- Reduced cost of development
- Higher quality software
- Encapsulation



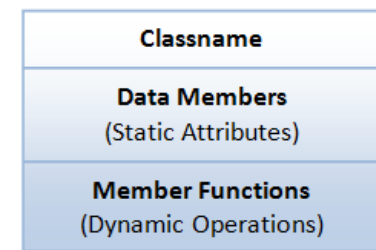
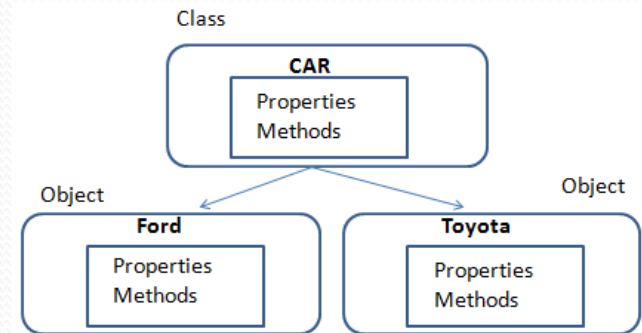
Object Oriented Concepts

- Class
- Object
- Inheritance
- Abstraction
- Encapsulation
- Polymorphism



Class – OO Concepts

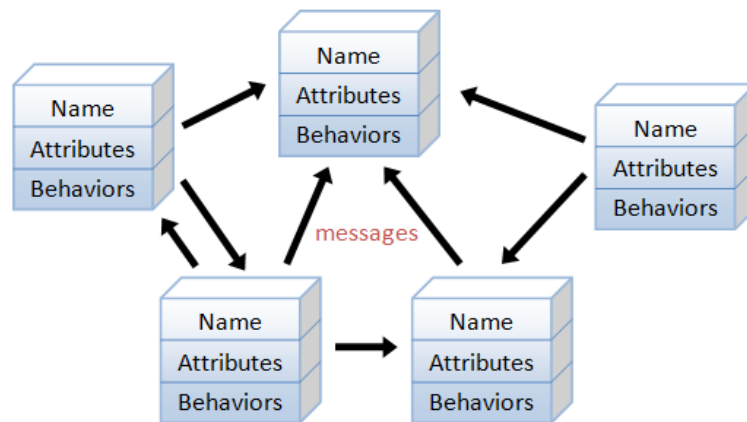
- The basic unit of OOP's is a 'class'.
- Class encapsulates both the *static attributes* and *dynamic behaviors* within a "box"
- Class combines "data structures" + "entity's algorithm"
- A class is a *blueprint* from which objects get created.
- Class defines the state and behavior, typically, of a real-world object



A class is a 3-compartment box encapsulating data and functions

Class – OO Concepts

- A class is a definition of objects of the same kind.
- Class is a blueprint, template, or prototype that defines and describes the *static attributes* and *dynamic behaviors* common to all objects of the same kind.



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Classname (Identifier)	Student	Circle
Data Member (Static attributes)	name grade	radius color
Member Functions (Dynamic Operations)	getName() printGrade()	getRadius() getArea()

SoccerPlayer	Car
name number xLocation yLocation	plateNumber xLocation yLocation speed
run() jump() kickBall()	move() park() accelerate()

Examples of classes

Object – OO Concepts

- An *object* is a realization of a particular item of a class.
- All instances (objects) of a class have same properties as in class definition
- Ex. Define a class called Student, create 3 instances: “Peter”, “Paul” & “Mary”.



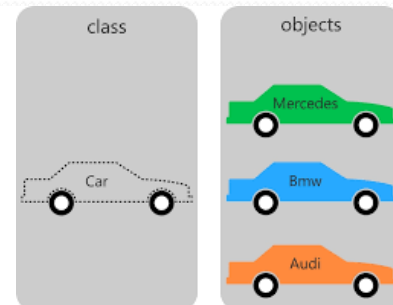
Objects

Class Definition

Circle
-radius:double=1.0
-color:String="red"
+getRadius():double
+getColor():String
+getArea():double

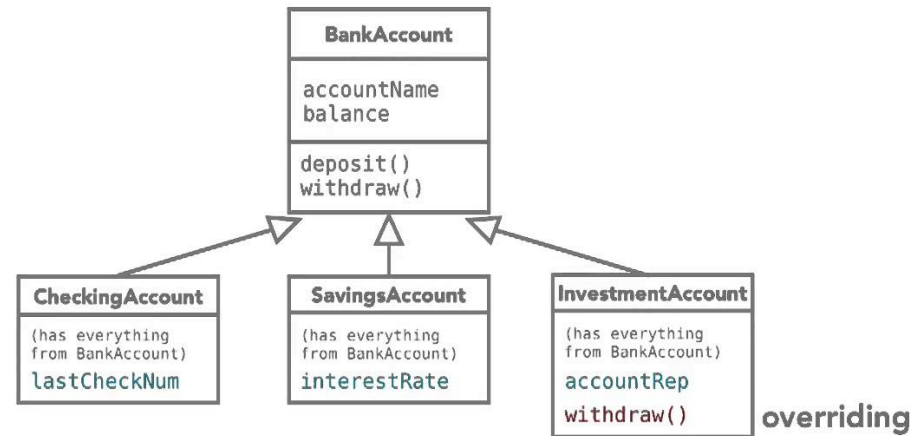
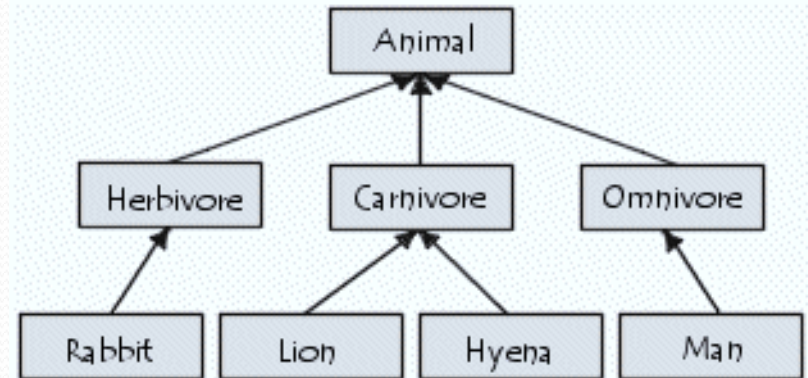
Instances

<u>c1:Circle</u>	<u>c2:Circle</u>	<u>c3:Circle</u>
-radius=2.0	-radius=2.0	-radius=1.0
-color="blue"	-color="red"	-color="red"
+getRadius()	+getRadius()	+getRadius()
+getColor()	+getColor()	+getColor()
+getArea()	+getArea()	+getArea()



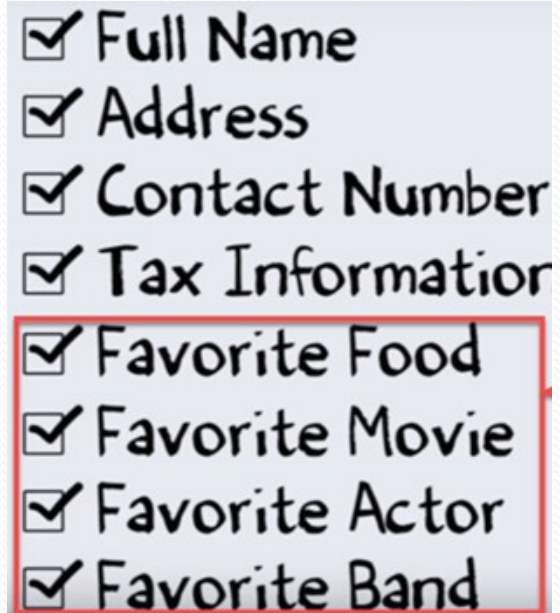
Inheritance – OO Concepts

- **Inheritance** enables new **objects** to take on the properties of existing **objects**.
- A class that is used as the basis for **inheritance** is called a superclass or base class.
- A class that inherits from a superclass is called a subclass or derived class.



Abstraction – OO Concepts

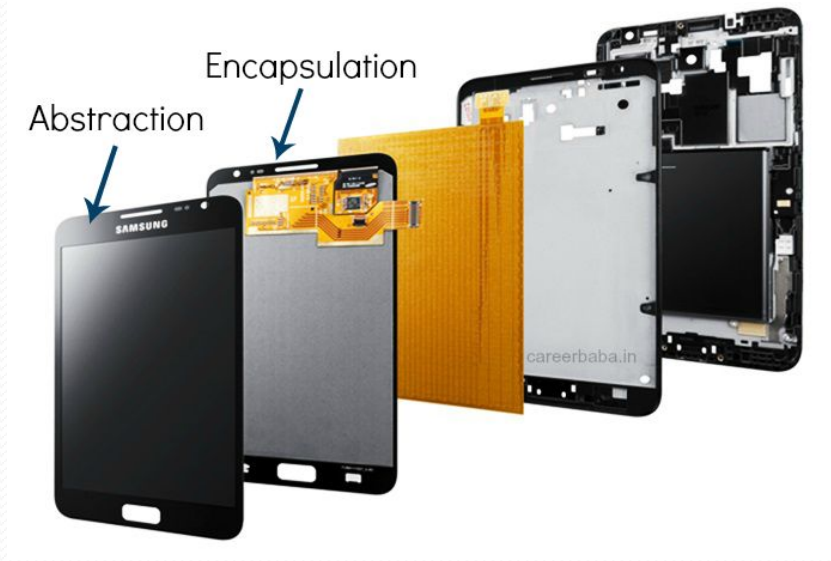
- Hide internal details
only show essential data about object to reduce complexity and increase efficiency
- Abstraction is selecting data from a larger pool to show only the relevant details of the object



<input checked="" type="checkbox"/>	Full Name
<input checked="" type="checkbox"/>	Address
<input checked="" type="checkbox"/>	Contact Number
<input checked="" type="checkbox"/>	Tax Information
<input checked="" type="checkbox"/>	Favorite Food
<input checked="" type="checkbox"/>	Favorite Movie
<input checked="" type="checkbox"/>	Favorite Actor
<input checked="" type="checkbox"/>	Favorite Band

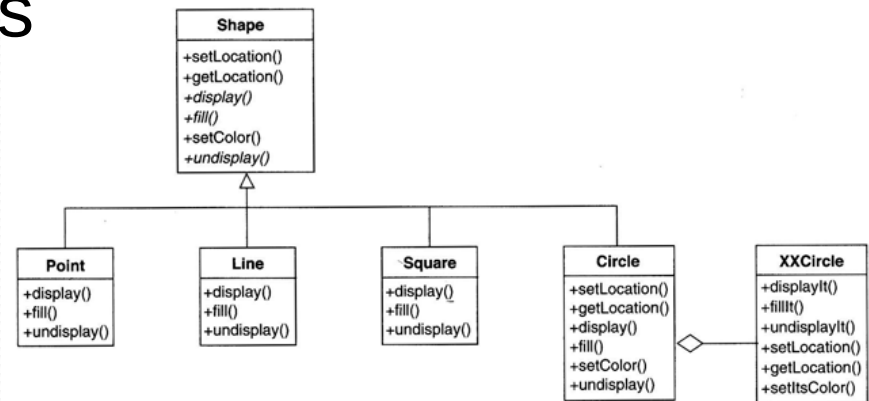
Encapsulation – OO Concepts

- Also called ‘data binding’
- Hide the state (value) of attributes within a class, preventing unauthorized direct access to it
- Control access to state of attributes + methods
- Bundling data and methods together in a class



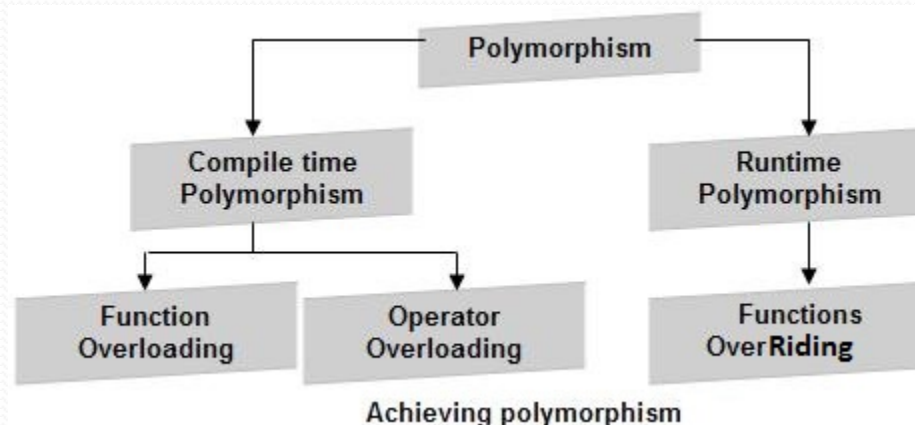
Polymorphism – OO Concepts

- Generally, the ability to appear in many forms
- Ability to process objects differently depending on their data types or class
- Single interface to entities of different types
- Instances (subclass) derived from superclass
- Static @ compile time
- Dynamic @ run time



Compile & Run-time Polymorphism

Compile-time Polymorphism	Run-time Polymorphism
Is implemented through method overloading .	Is implemented through method overriding .
Is executed at the compile-time since the compiler knows which method to execute depending on the number of parameters and their data types.	Is executed at run-time since the compiler does not know the method to be executed, whether it is the base class method that will be called or the derived class method.
Is referred to as static polymorphism.	Is referred to as dynamic polymorphism.



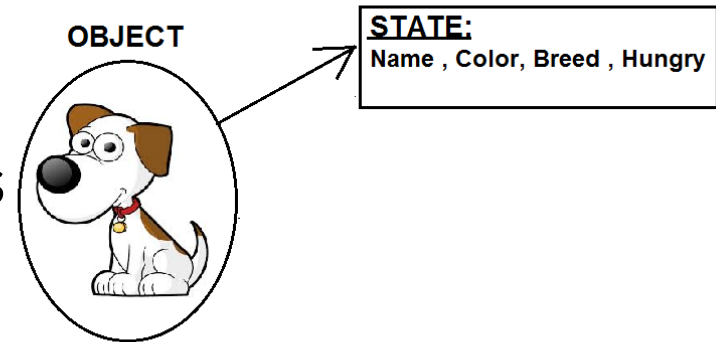
Instance – OO Concepts

- An 'instance' is a unique copy of a class, that represents an object.
- When a new instance of a class is created, the JVM will allocate space in memory for that class instance.
- 'Instance' and 'object' can be used interchangeably



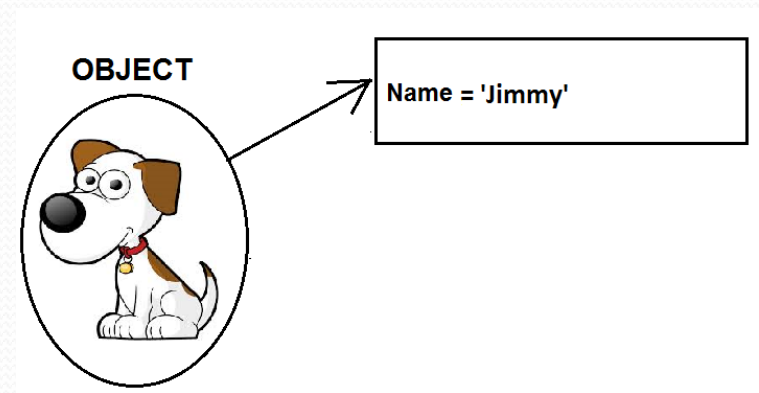
State – OO Concepts

- An **object** has **state** (data) and behavior (functions).
- Is the instances (variables) inside the *object* that contains the data.
- State is the values of its attributes.



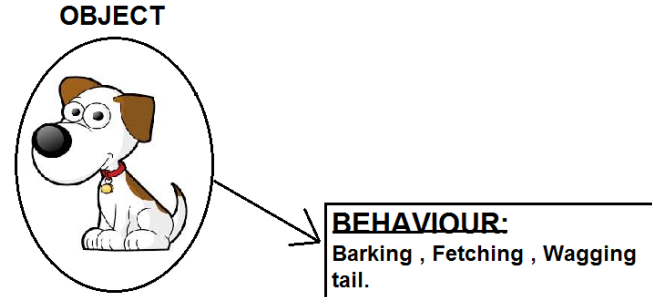
Attribute – OO Concepts

- An Attribute is a named property of a class.
- It has a type.
- It describes the range of values that this property may hold.
- A object can contain any number of attributes



Method (Operation) – OO Concepts

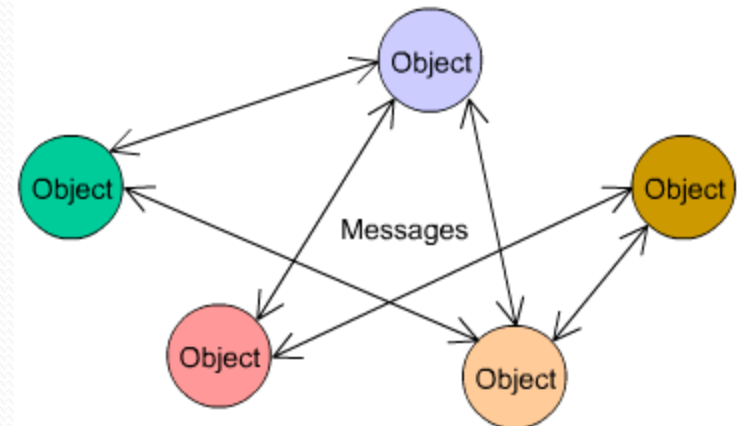
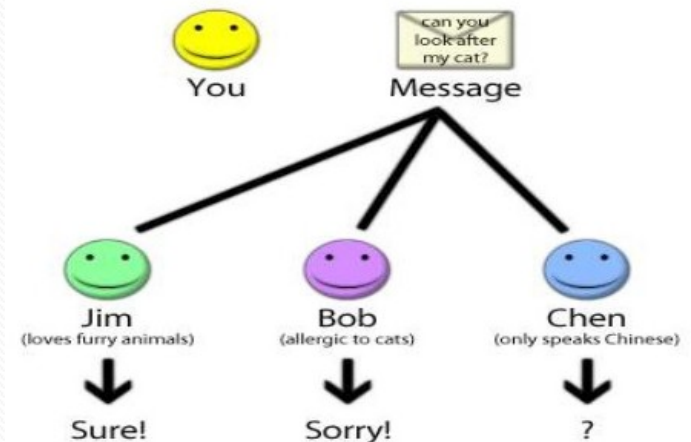
- The behaviour of an object is defined by its methods, which are the functions and subroutines defined within the object **class**.
- An **operation** that an object can perform.



Message Passing – OO Concepts

- Message passing is the communication between processes.
- Message passing is a form of communication used in object-oriented programming.
- Communications are completed by the sending of messages (functions and data) to recipients.

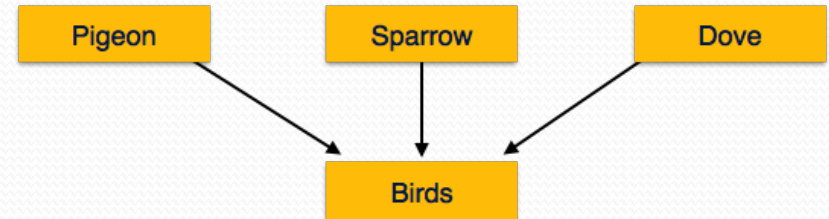
Message Passing



Interaction of objects via message passing

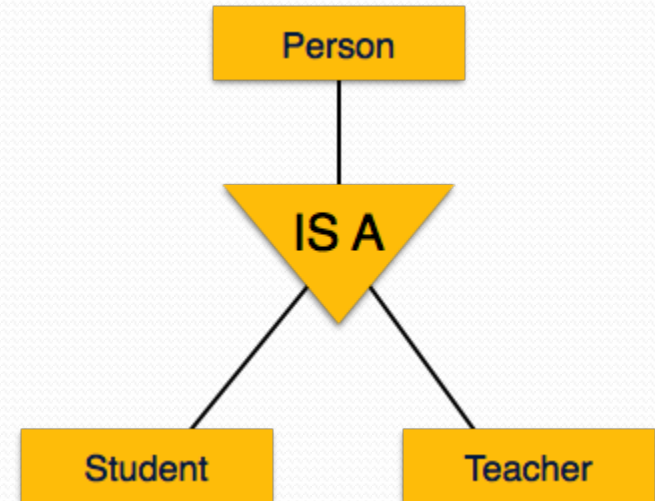
Generalization – OO Concepts

- The generalized entities contain the properties of all the generalized entities
- Also called as the 'superclass'
- Example: Pigeon, House sparrow, Crow and Dove can all be generalized as 'Birds'.



Specialization – OO Concepts

- Specialization is the opposite of generalization.
- In specialization, a group of entities is divided into sub-groups based on their characteristics.
- For example: A 'Person' has name, date of birth, gender, etc. These properties are common in all persons, human beings.
- But in a company, persons can be identified as employee, employer, customer, or vendor, based on what role they play in the company.



History of Java



- 1990: 'Oak'
 - To control microprocessors, embedded in customer products
- Oak needed to be:
 - Platform Independent
 - Extremely reliable
 - Compact
- 1993: 'Java'
 - Internet and web exploration
 - Internet applications
- 1995: 'Java 1.0'



James Gosling:
Inventor of Java

Features of Java

- **Compile & Interpret:** Java compiler converts Java Source code into Java bytecode. Java interpreter converts bytecode into machine code
- **Portable & Platform Independent:** Java programs are easily transferred from one computer to another, bytecode interpreted on JVM
- **Object Oriented:** Java is 100% object oriented language
- **Robust:** Java supports automatic memory management and exception handling.
- **Distributed:** Has the ability to share & access data remotely

Features of Java

- **Simple:** Easy to understand syntax, doesn't support pointers, operator overloading, multiple inheritance
- **Multithreaded:** Java has the ability to write programs that do multiple tasks simultaneously
- **Dynamic:** Java allocates memory at runtime for classes, methods and objects as and when required

References

- <https://www.javatpoint.com/java-oops-concepts>
- <https://beginnersbook.com/2013/04/oops-concepts/>
- <https://www.guru99.com/java-oops-concept.html>
- <https://www.journaldev.com/12496/oops-concepts-java-example>



Happy Oops Programming!!
Thank you!