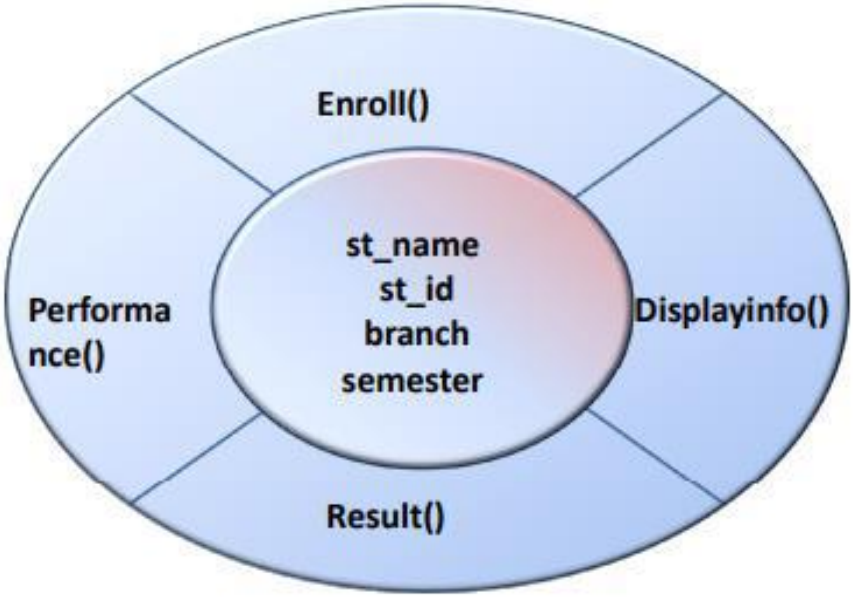
Basic principles of Object Orientation- Class, Object, Abstraction, Encapsulation,inheritance,polymorphism

Class

- **Class is a collection of similar objects.**
- Class = **template or blueprint that describe behavior**
- A class is created with **data members and member function**
- Once class is defined **any number of object belonging to this class can be created**
- Syntax:

```
class class_name
 {
   Attributes; //Properties(instance variables)
   Operations; //Behaviours (methods)
}
```

Example: StudentObject



Example :class student

```
 {
char st_name[30];
char st_id[10];
char branch[10], semester[10];
Void Enroll( );
Void Displayinfo( );
Voide Result( );
Void Performance( );
}
```

object

- OOP uses objects as its **fundamental building blocks.**

- Objects are the **basic run-time entities in an object-oriented system.**

- Object is a real world existing entity.

- Real world entity means we can use the properties and behavior of an Object .

- Every object is associated with data and functions which define meaningful operations on that object.

➢ Object is an **Instance of a particular class.**

➢ All the objects have **a state, behaviour and identity.**

State of an object -

❖The state or **attributes are the built in characteristics or properties of an object**

❖For example, Student have a **first name, last name, age, etc.**

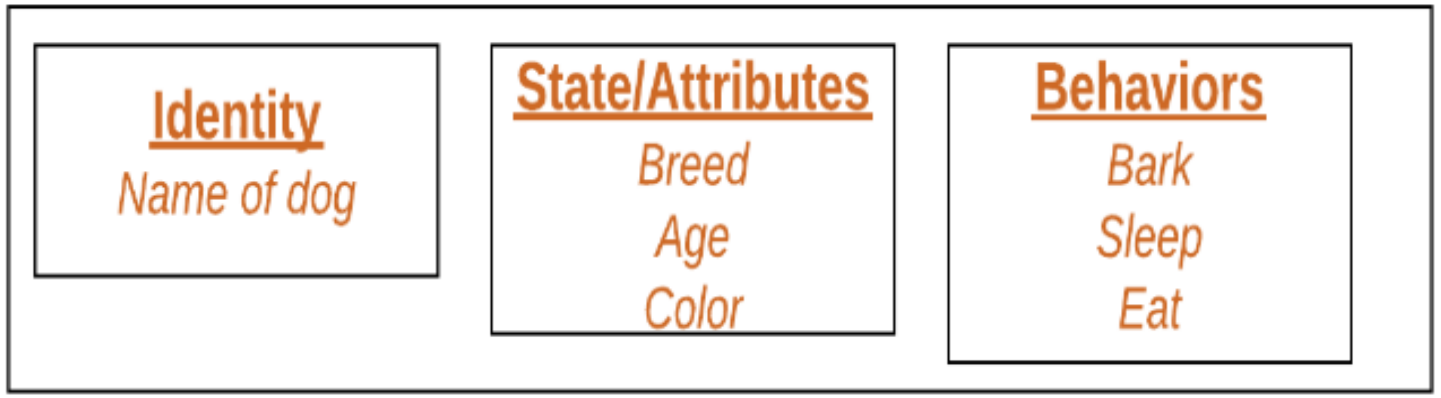
Behaviour of the object -

• It is represented by methods of an object. It also reflects the response of an object with other objects.

Object identity -

• It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog



Abstraction

- A process of hiding the internal implementation details and highlighting only the set of services/functionality to the user is called Abstraction .

- **Example : Bank ATM Screens (Hiding their internal implementation and highlighting set of services like withdraw, money transfer, mobile registration).**

- In java, we use **abstract class and interface to achieve abstraction.**

Abstraction in Java

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Interface or AbstractClass

SMSSender

sendSMS(String message);

SMSSenderImpl

```
sendSMS(String message)
{
    // Logic to send a SMS.
}
```

- ✓ **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- ✓ Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.
- ✓ Abstraction lets you focus on what the object does instead of how it does it.

Encapsulation

- The process of **binding(or wrapping) the data and the code(or function) into a single unit is called Encapsulation.**

- Every Java class is the example of encapsulation.

- Java bean is the fully encapsulated class because all the data members are private here.

- This is the practice of keeping fields within a class **private, then providing access to them via public methods.**

- In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

Class

Methods

Public Getter & Setter
(Visible to all)

Variables

Private variables
(Not visible)

Encapsulation

Binding of data
(methods +
variables)
into single entity

Inheritance

- Inheritance is the process of acquiring properties and behaviours from one **object to another object or one class to another class.**

- In inheritance, we derive a new class from the existing class. Here, the new class acquires the properties and behaviors from the existing class.

- In the inheritance concept, the class which provides properties is called as **parent class and the class which receives the properties is called as child class.**

- The parent class is also known as **base class or super class.The child class is also known as derived class or sub class.**

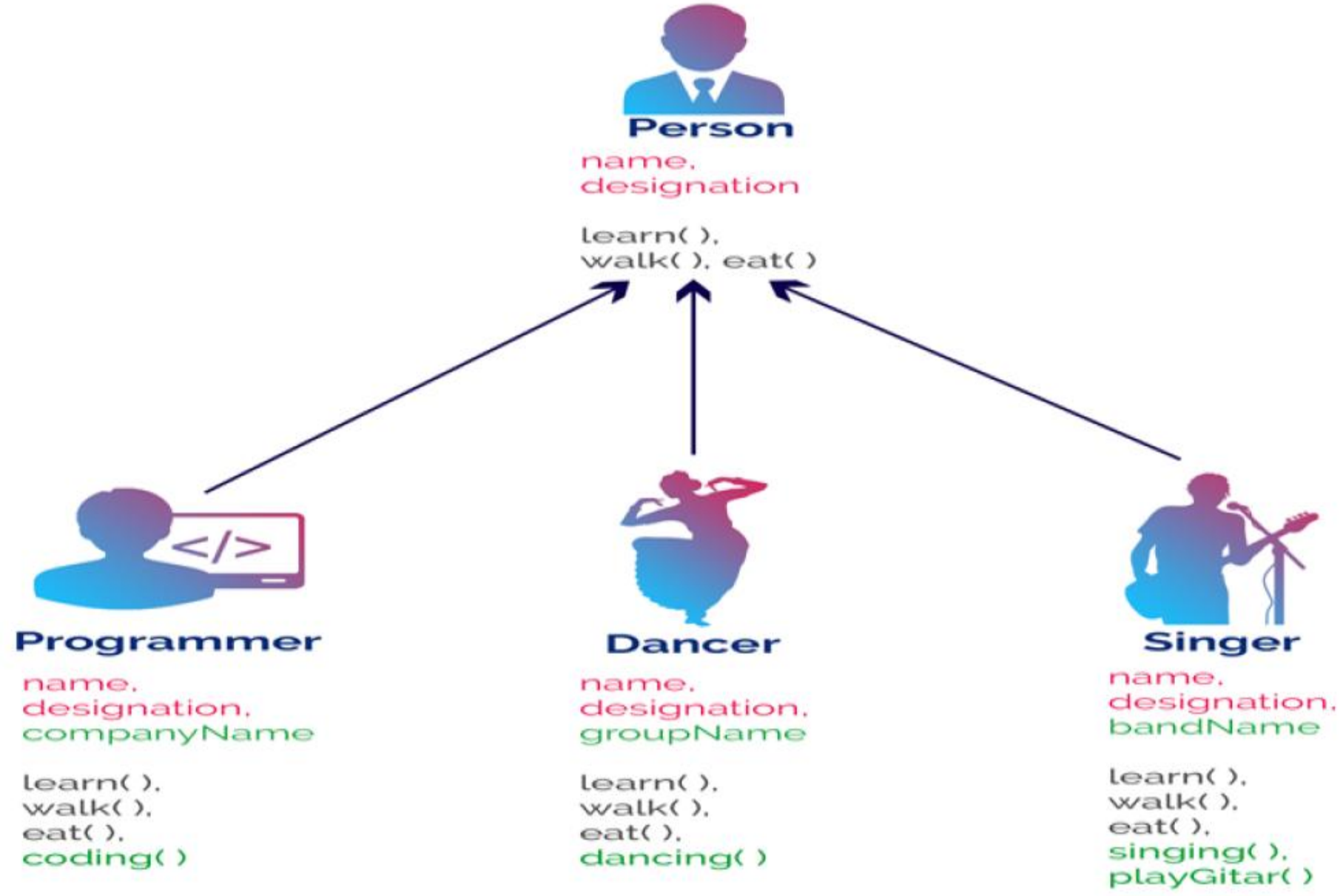
- In the inheritance, the properties and behaviours of base class extended to its derived class, but the base class never receive properties or behaviours from its derived class.

- In java programming language the keyword **extends is used to implement inheritance.**

- In java, using the inheritance **concept, we can use the existing features of one class in another class.**

- The inheritance provides a great advantage called **code re-usability. With the help of code re-usability, the commonly used code in an application need not be written again and again.**

- It is used to **achieve runtime polymorphism.**


Person
name,
designation
learn( ),
walk( ), eat( )
Programmer
name,
designation,
companyName
learn( ),
walk( ),
eat( ),
coding( )
Dancer
name,
designation,
groupName
learn( ),
walk( ),
eat( ),
dancing( )
Singer
name,
designation,
bandName
learn( ),
walk( ),
eat( ),
singing( ),
playGitar( )

Polymorphism

- The **Method Overloading(**or Ad hoc polymorphism**) is a technique used to define the same method with different implementations and different arguments.**

- The **Method Overriding**(or pure polymorphism) is a technique used to define the **same method with the same arguments but different implementations.**

Person

learn( )

Same method with different implementations

Programmer

learn( )

with coding languages

Dancer

learn( )

with dancing steps

Singer

learn( )

with tunes, music, songs

Modularity

- Modularity is **the process of decomposing a problem (program) into a set of modules so as to reduce the overall complexity of the problem**.

- **Modularity is the degree to which a system's components are made up of relatively independent components or parts which can be combined.**

- Modularity refers to the concept of making multiple modules first and then linking and combining them to form a complete system.
- Modularity enables **re-usability and minimizes duplication**

- In addition to re-usability, modularity also makes it easier to fix problems as bugs can be traced to specific system modules, thus limiting the scope of detailed error searching.

- The technique of breaking down one big solution into smaller modules for ease of development, implementation, modification and maintenance is called **modular technique of programming or software development.**

- Modular programming is an extensively used concept based on modularity.

- The figure shows modules of a puzzle which can form different shapes when they are placed at different places. The modules can be moved freely without affecting the functionality of other modules but it changes the system's shape(functionality).

Message passing

- Message passing is a form of communication between objects, processes or other resources used in object-oriented programming, inter-process communication and parallel computing.

- In this model, processes or objects can send and receive messages (signals, functions, complex data structures, or data packets) to other processes or objects.

- The concept of message passing makes it easier to build systems that model or simulate **real-world problems.**

- Message passing in Java is like sending an object i.e. message from one thread to another thread. It is used when threads do not have shared memory and are unable to share monitors or semaphores or any other shared variables to communicate.

Message Passing in Java