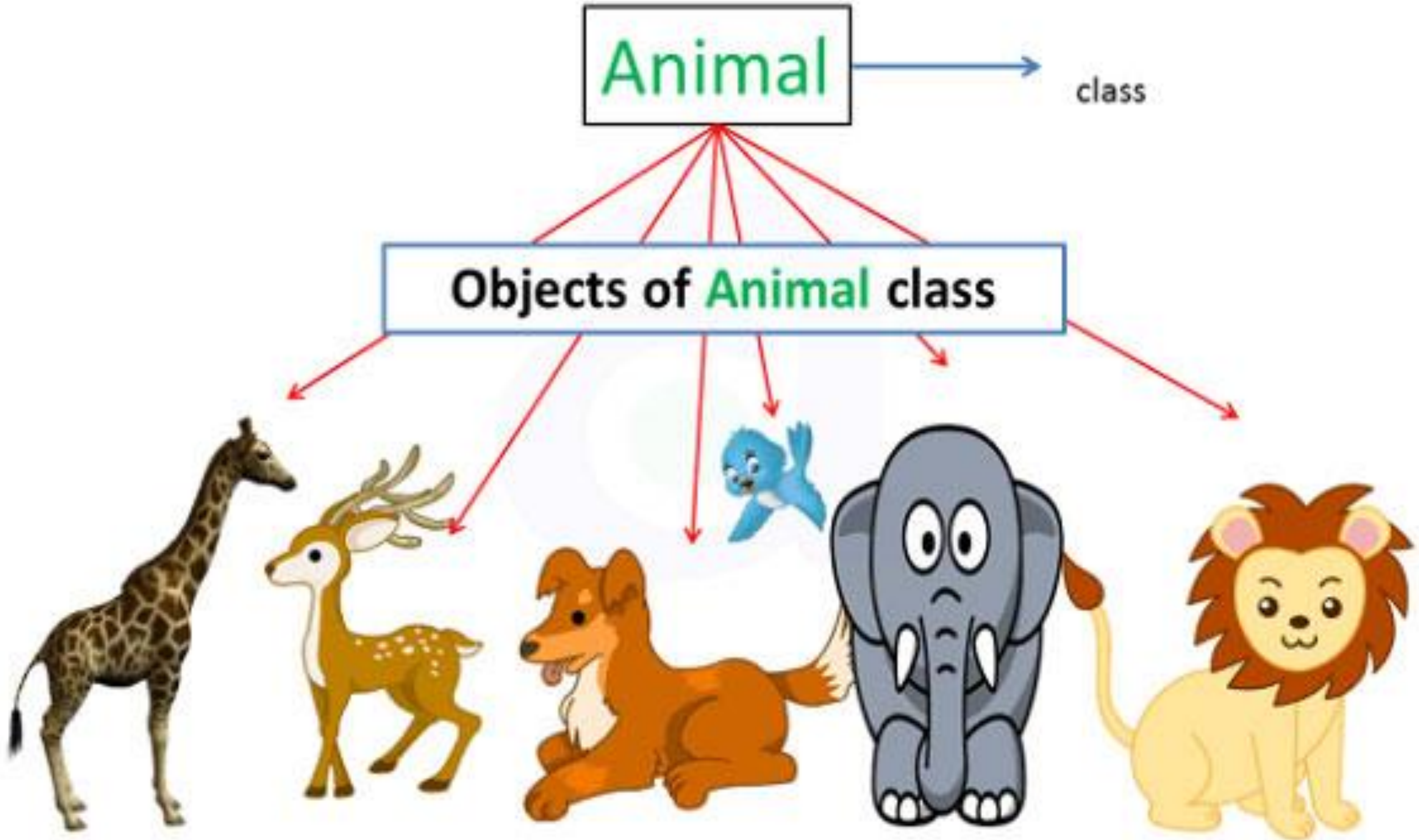
Object Oriented Programming

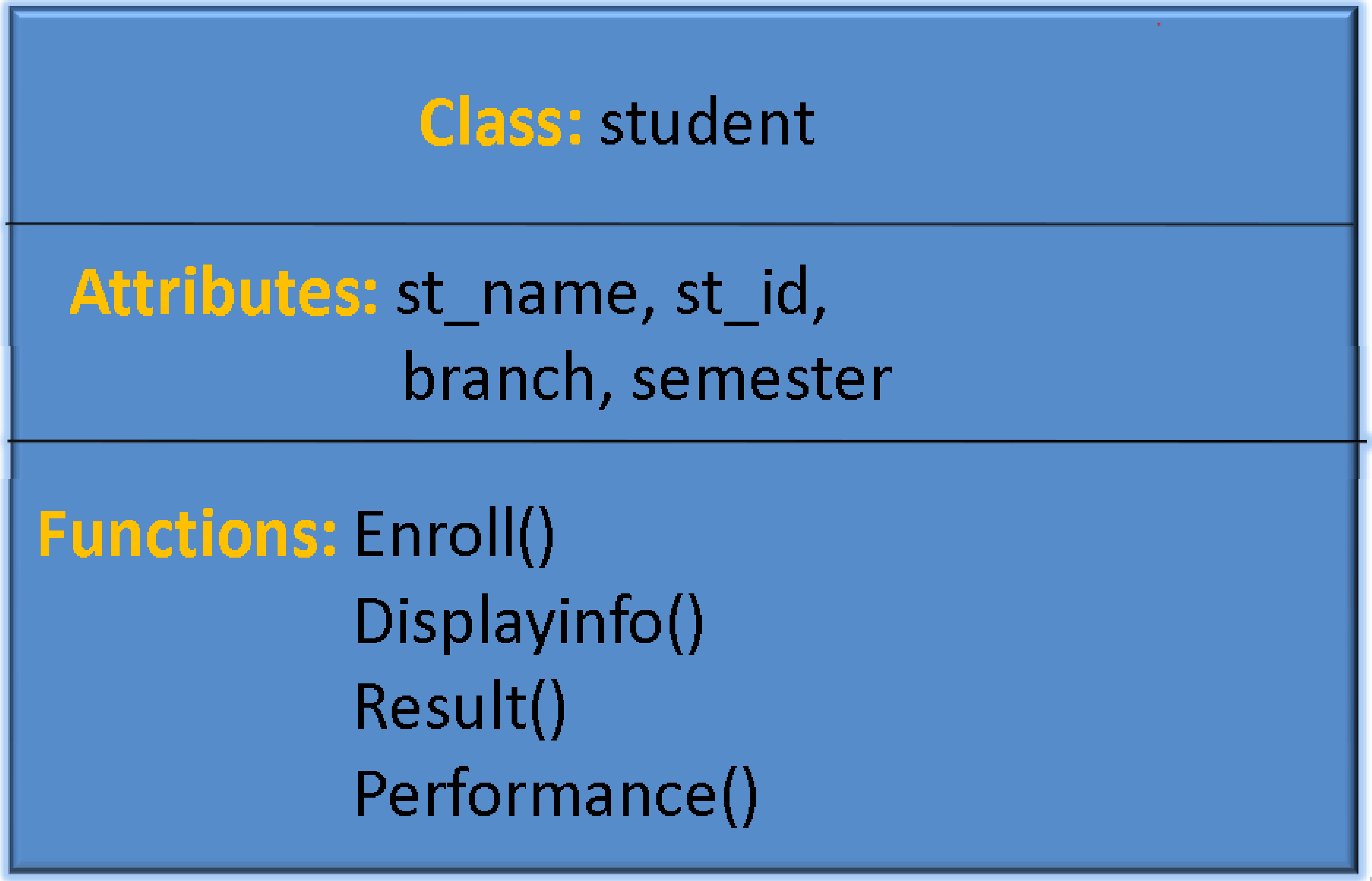
CLASS and OBJECT

Almost everything in the world can be represented as an object

- A flower, a tree, an animal
- A student, a professor
- A desk, a chair, a classroom, a building
- A university, a city, a country
- The world, the universe
- A subject such as CS, IS, Math, History, …
- An information system, financial, legal, etc..


Animal
class
Objects of Animal class

```
class class_name
{
     Attributes;        //Properties
    Operations;      //Behaviors
}
```

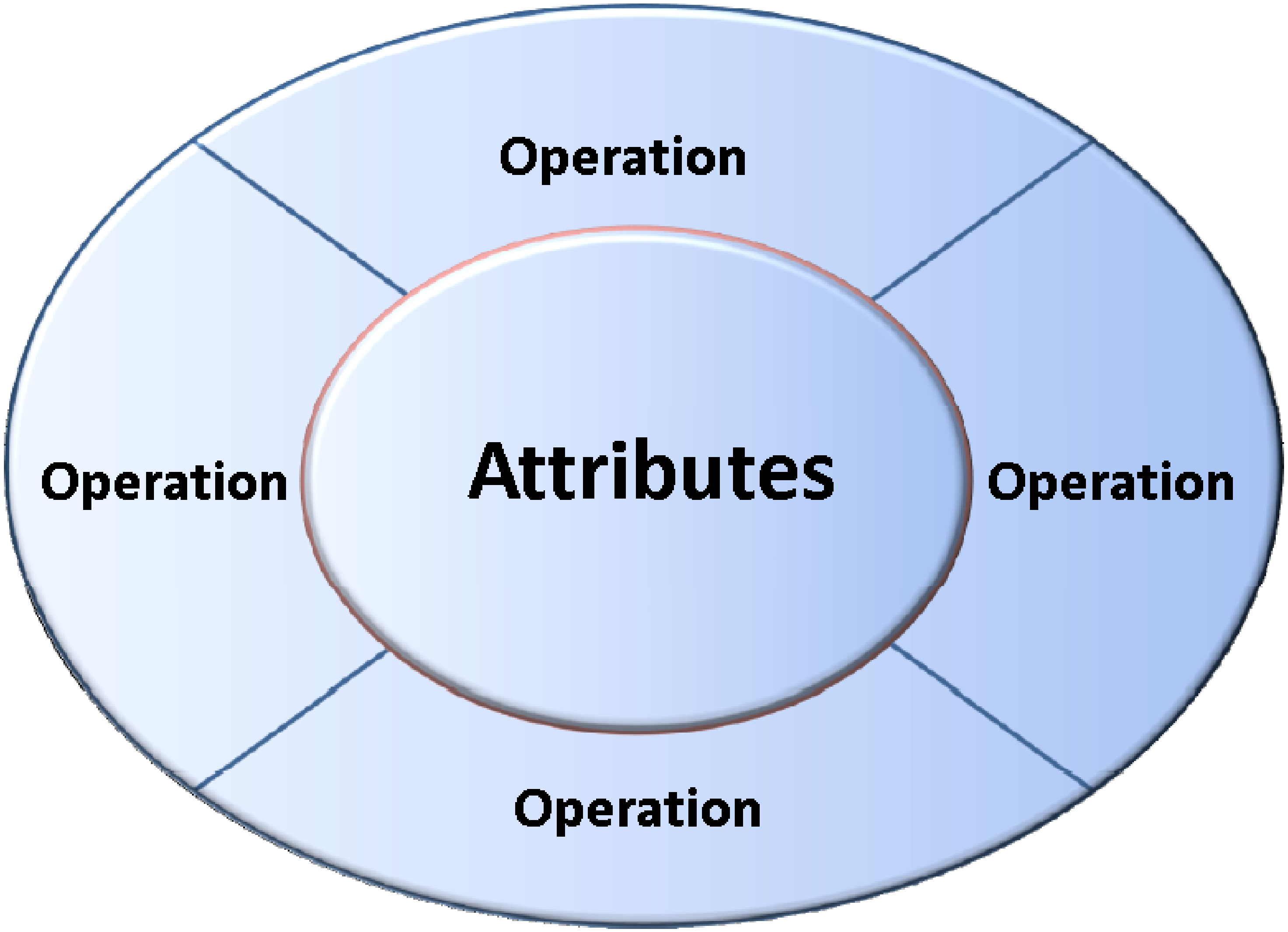


Class: student

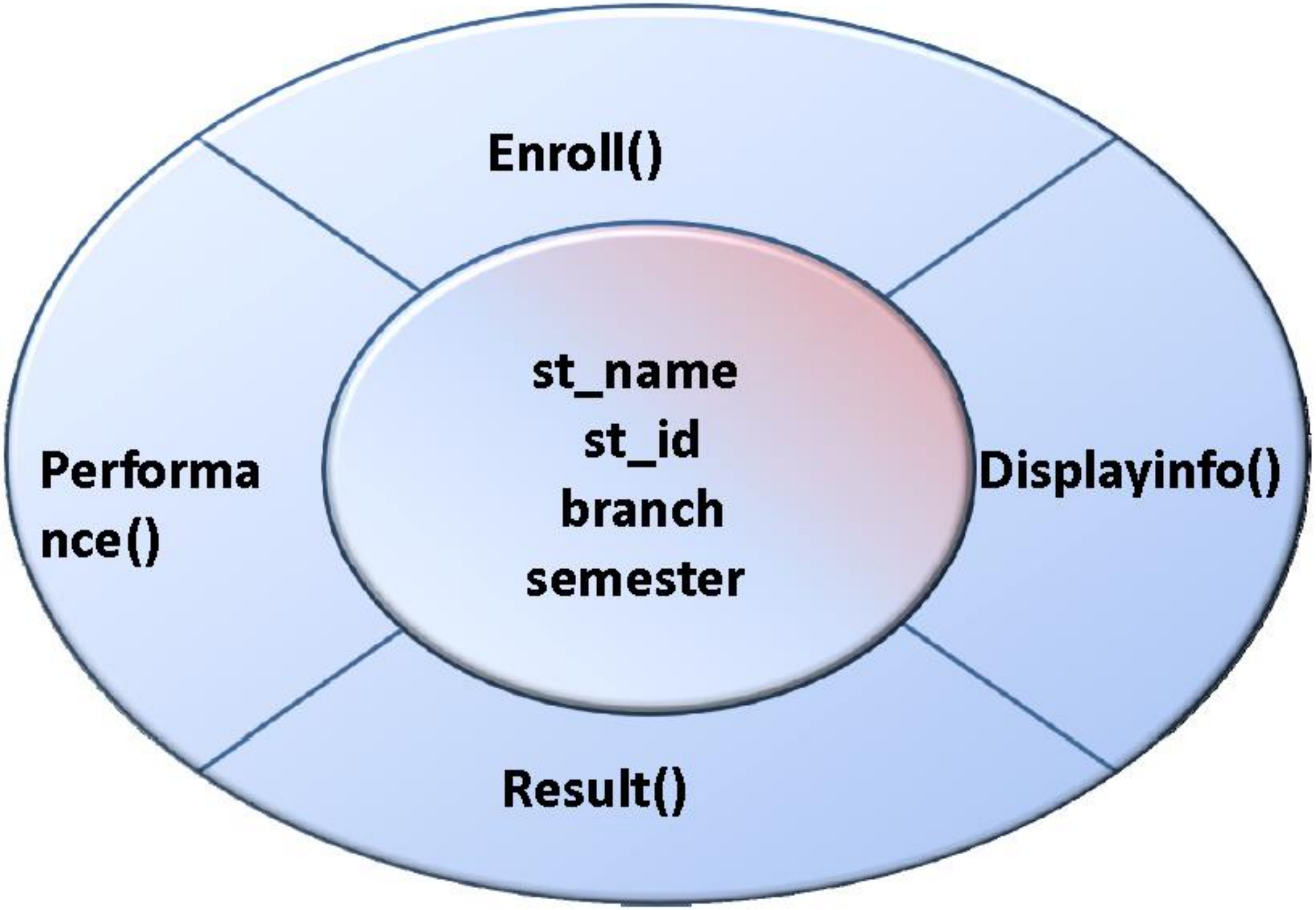
Attributes: st_name, st_id, branch, semester

Functions: Enroll()
Displayinfo()
Result()
Performance()

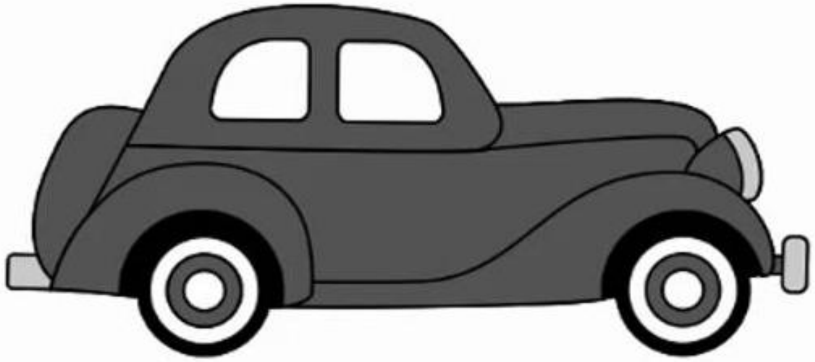
Object



Example: StudentObject



Properties
Make
Model
Color
Year
Price



Events
On_Start
On_Parked
On_Brake

Methods
Start
Drive
Park

Object

In C#, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.
In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.
Object is a runtime entity, it is created at runtime.
Object is an instance of a class. All the members of the class can be accessed through object.
Let's see an example to create object using new keyword.

```
Student  s1 = new  Student();      //creating an object of Student
```

In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class. The new keyword allocates memory at runtime.

Class

In C#, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Classes and objects are the two main aspects of object-oriented programming

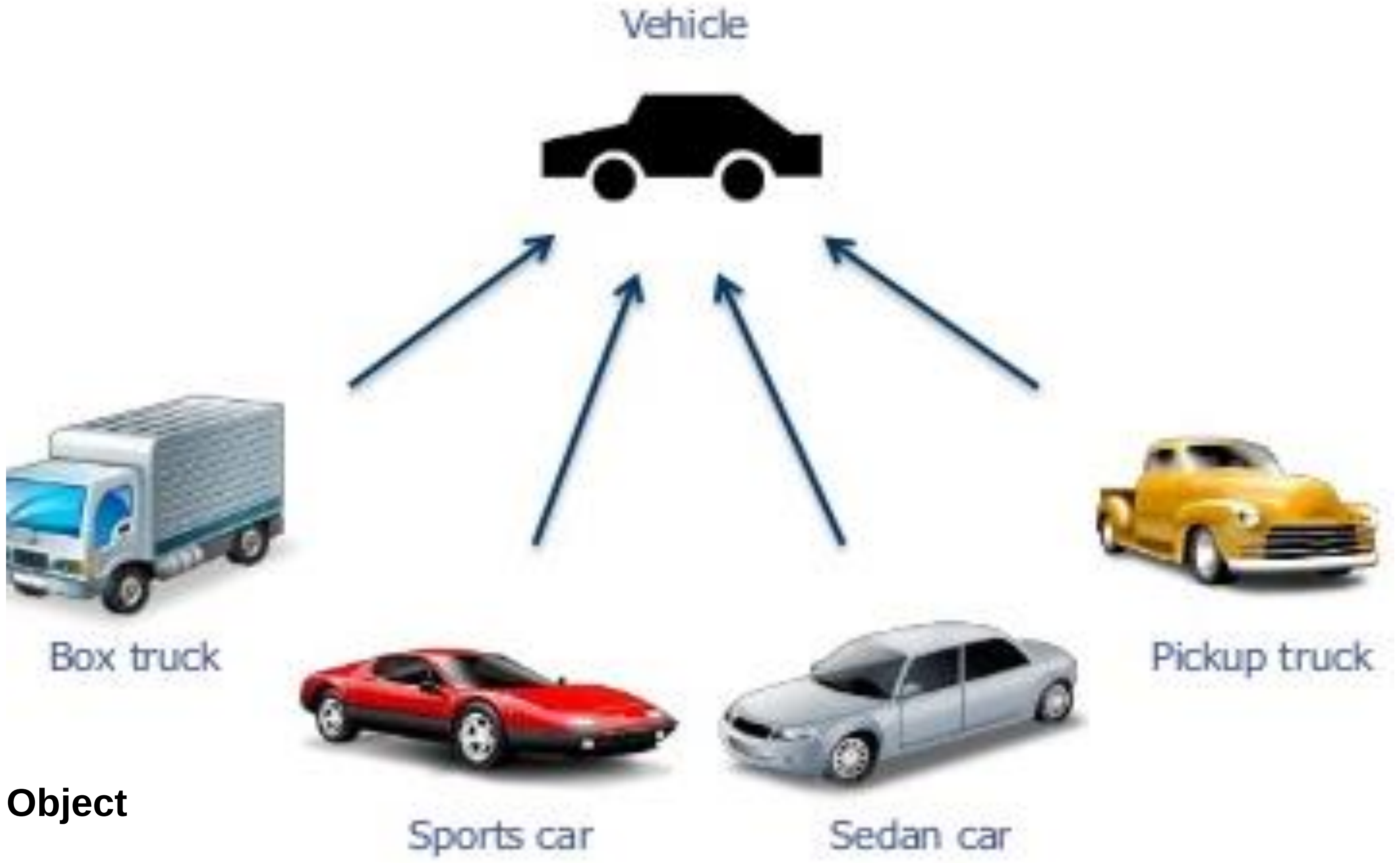
- **Class** – A **class** is a data type that allows programmers to create objects. A class provides a definition for an object, describing an object's attributes (data) and methods (operations).

- **Object** – Unique programming entity that has *methods*, has *attributes* and can react to *events*. An **object** is an *instance* of a class. **Objects** are the basic run time entities in an object-oriented system.

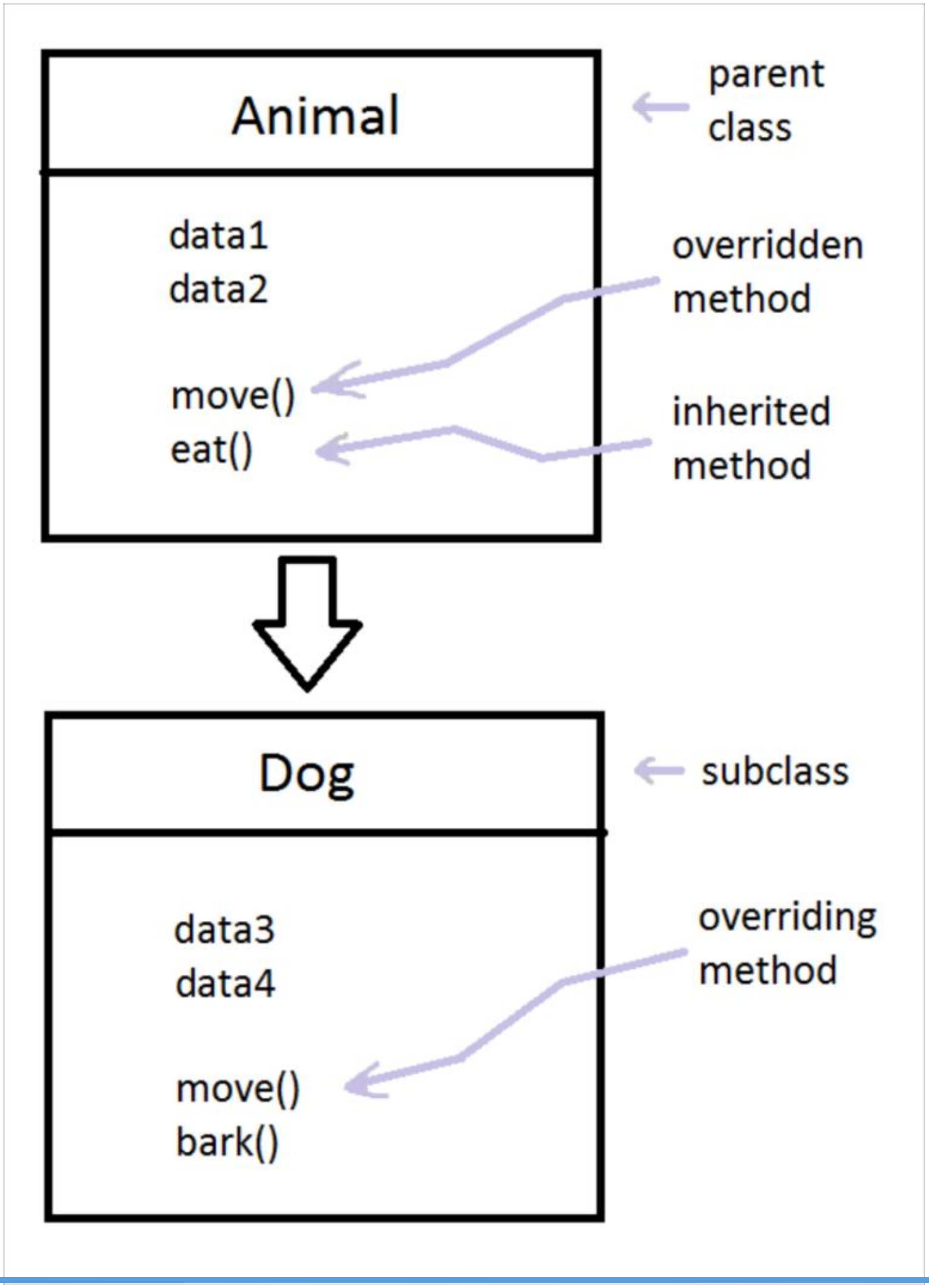
- **Attribute** – Things which describe an object; the "adjectives" of objects. In code, usually can be identified by a "descriptive" word – *Color, Name* (The attributes are some time called data members because they hold information.

- **Method** – Things which an object can do; the "verbs" of objects. In code, usually can be identified by an "action" like *Hide*, *Show* (sometimes called methods or member function*)*

Class



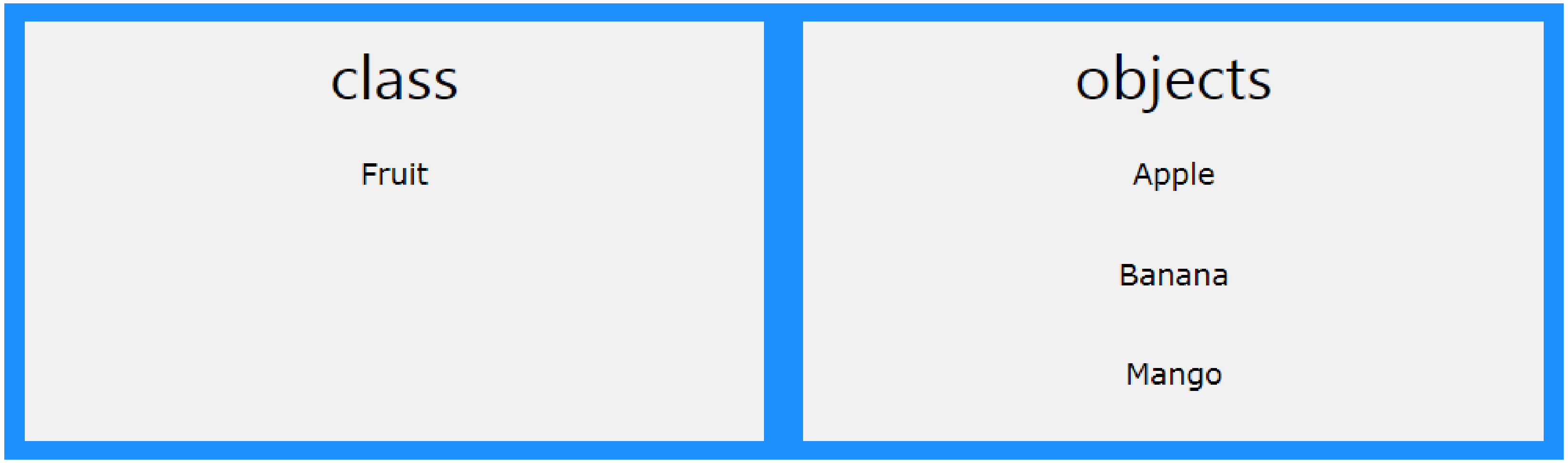
Object



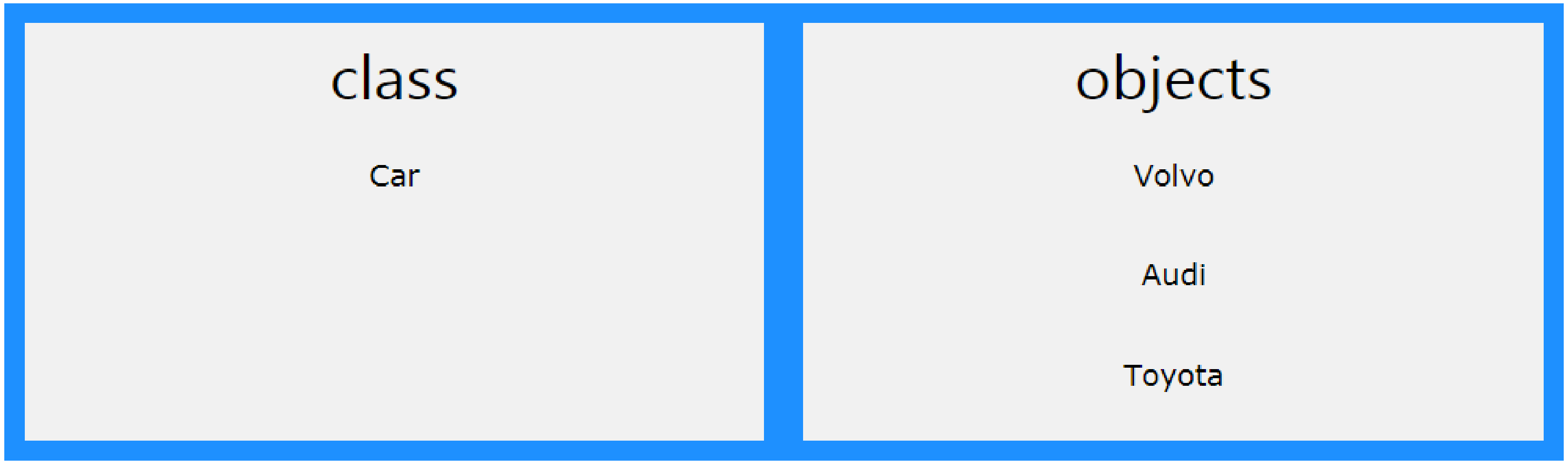
Class

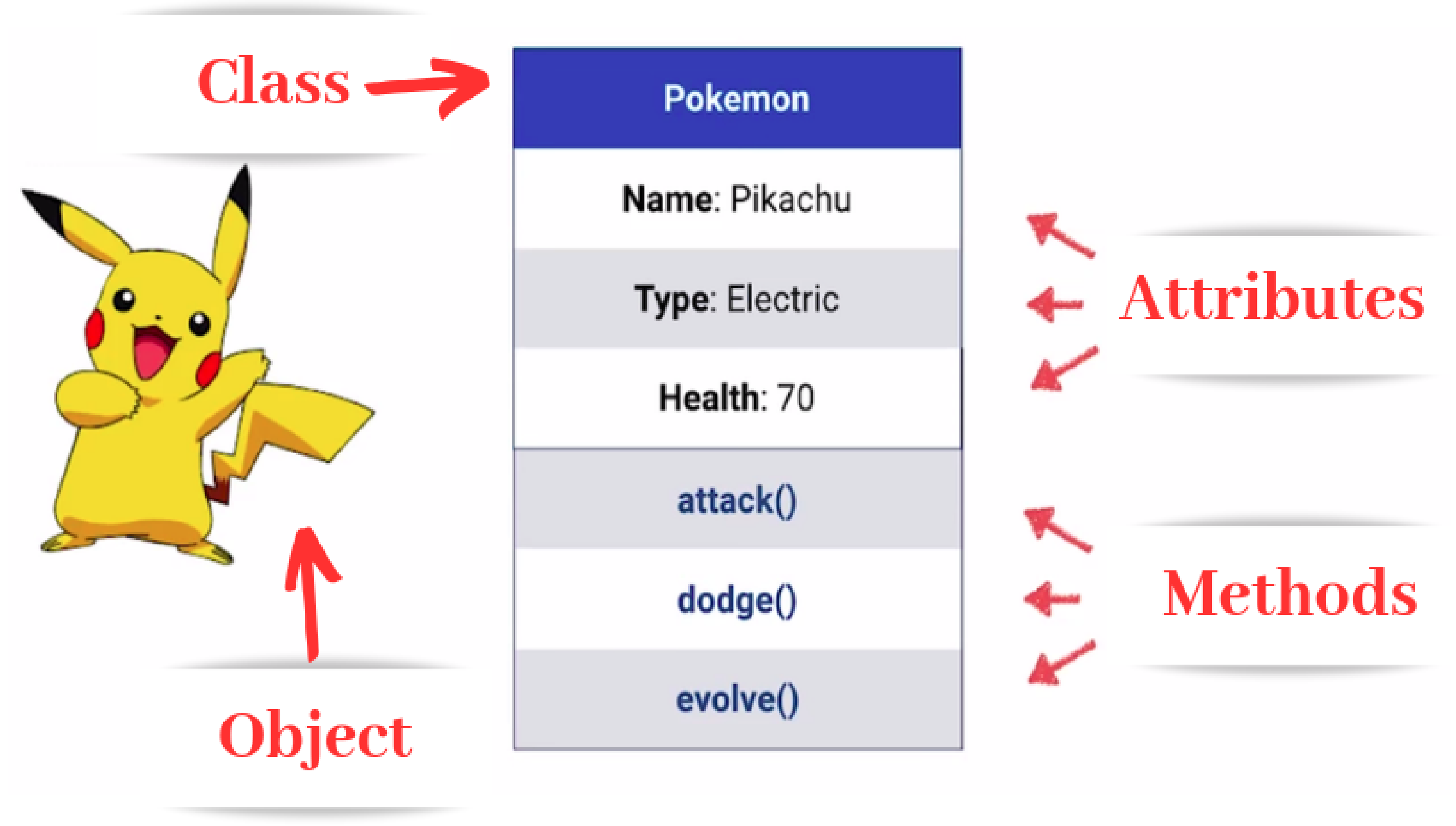
Object

a class is a template for objects, and an object is an instance of a class.
When the individual objects are created, they inherit all the variables and methods from the class.



Another example:





Class **Pokemon : Pikachu** **Object**

OBJECTS: STUDENT

DATA

- Name
- Date-of-birth
- Marks

Attributes

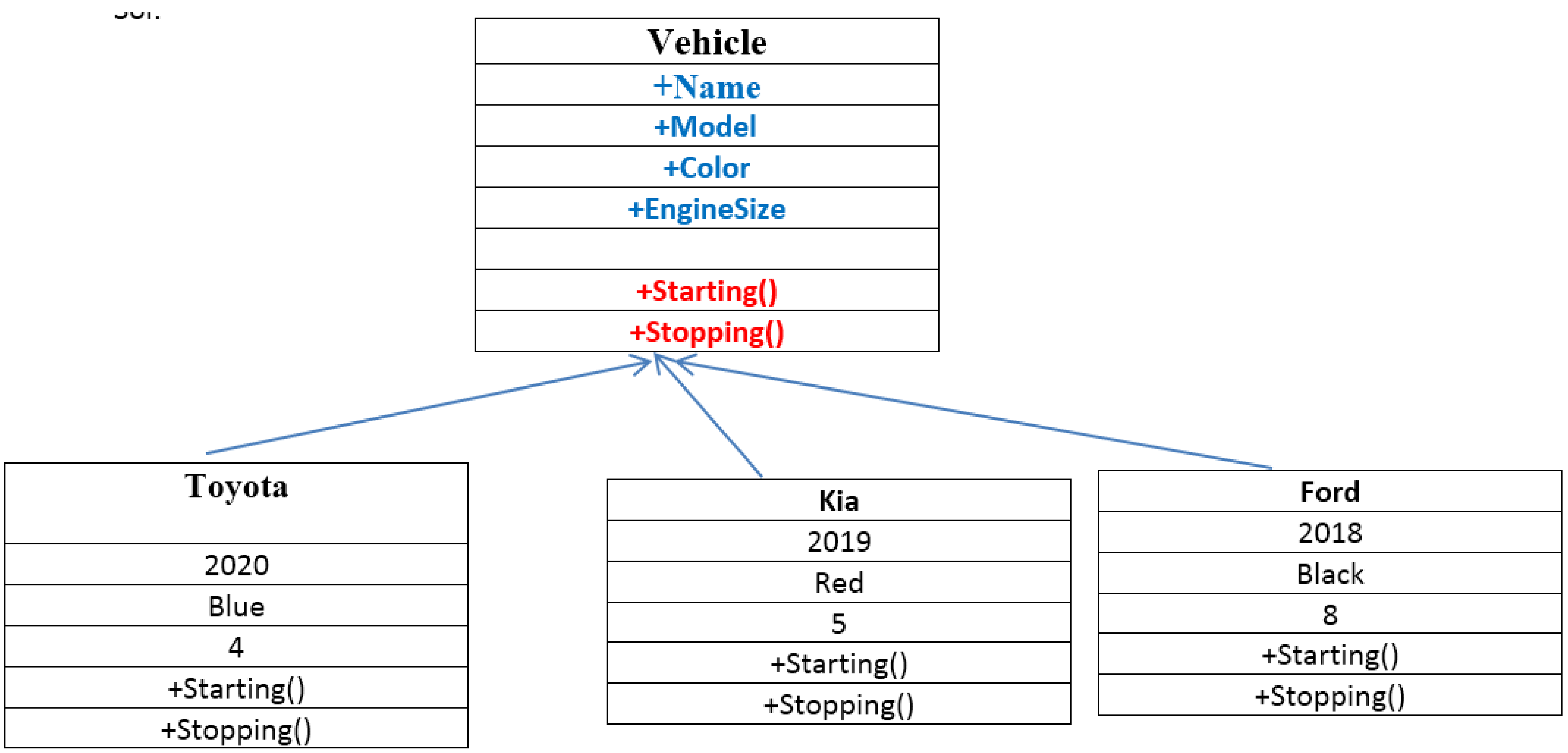
FUNCTIONS

- Total
- Average
- Display
- ………

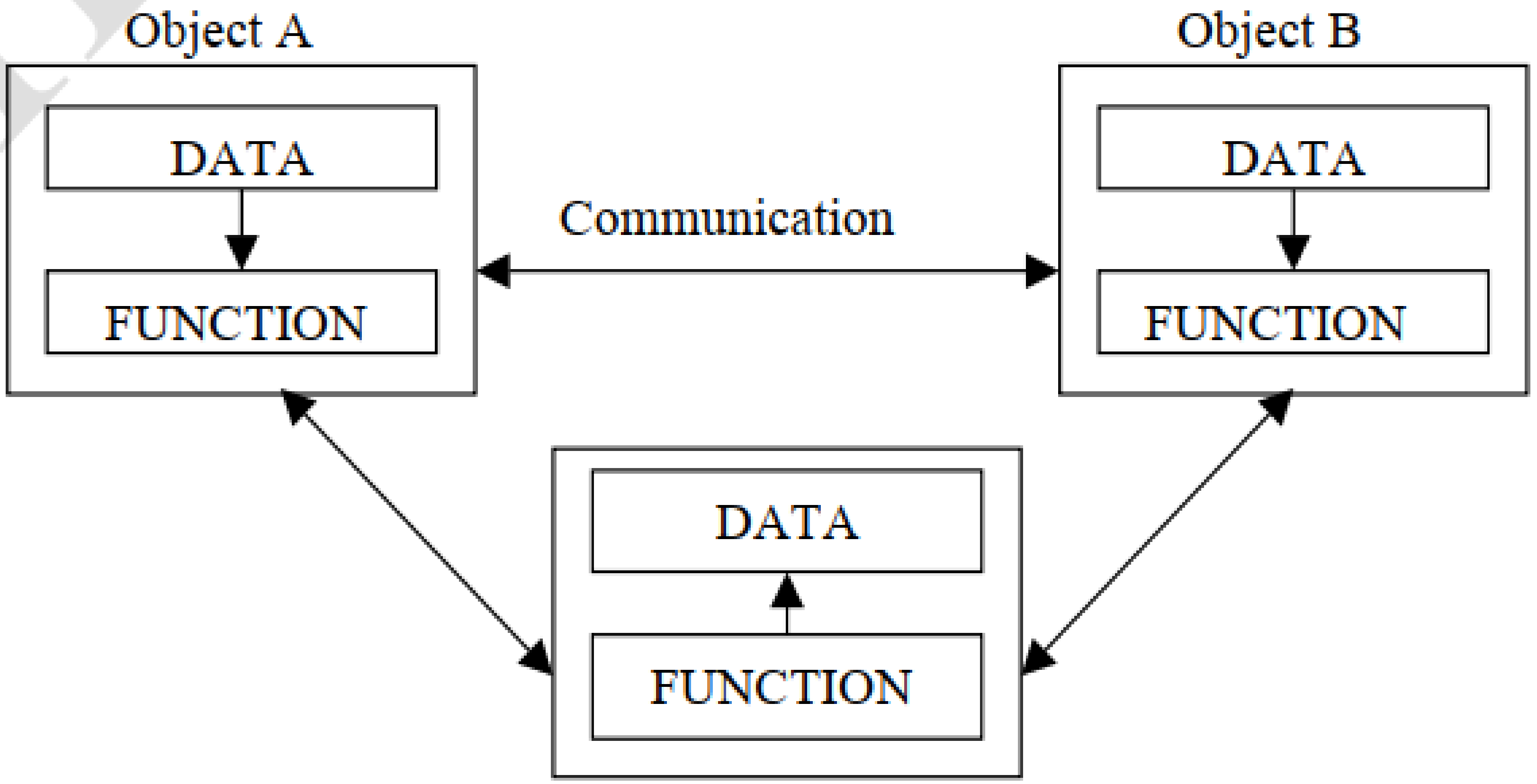
Methods

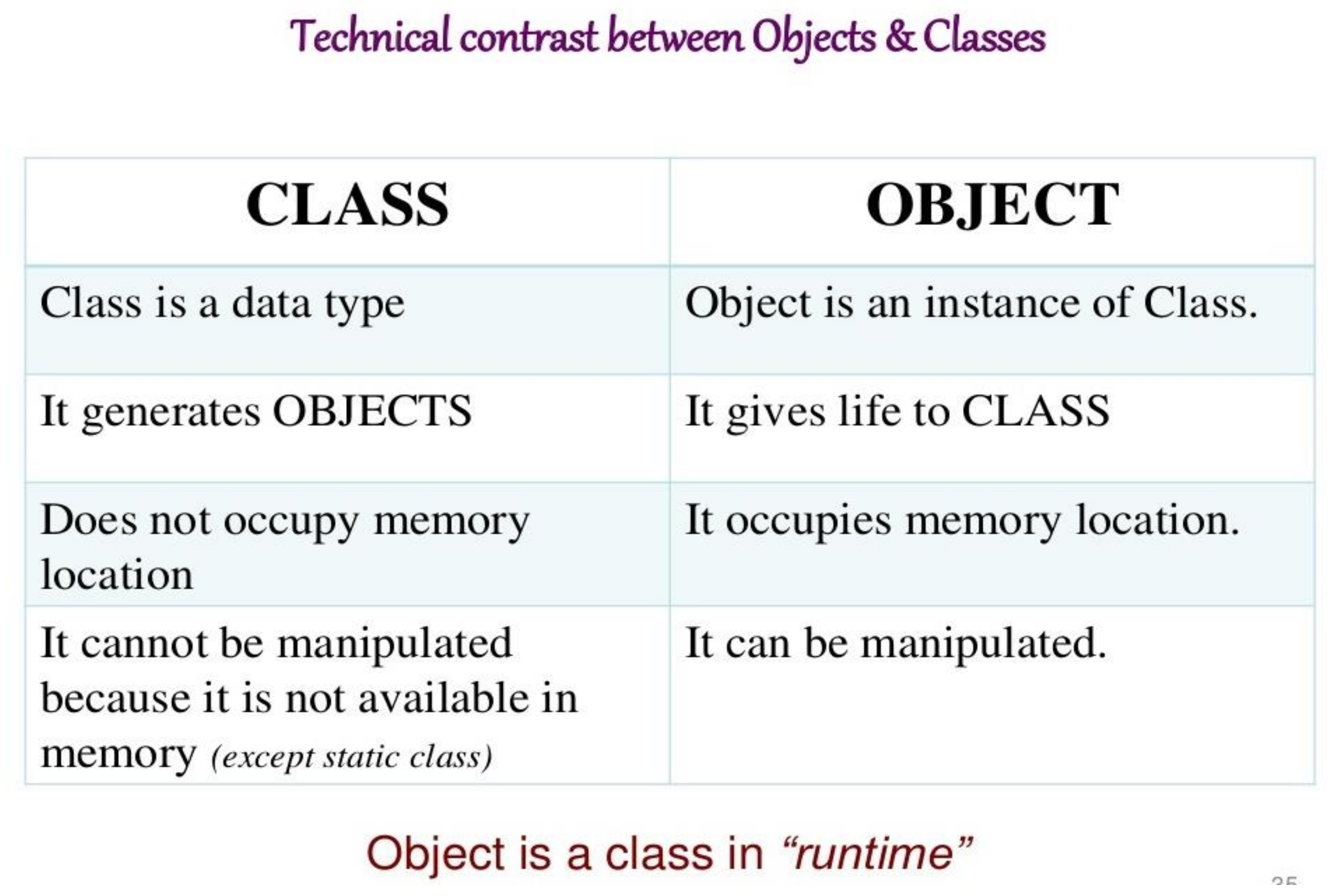
UML representation

The **Unified Modeling Language** (**UML**) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.



Organization of data and function in OOP





Technical contrast between Objects & Classes

CLASS	OBJECT
Class is a data type	Object is an instance of Class.
It generates OBJECTS	It gives life to CLASS
Does not occupy memory location	It occupies memory location.
It cannot be manipulated because it is not available in memory *(except static class)*	It can be manipulated.

Object is a class in *"runtime"*

Here is the syntax and declaration example of class

```
 class Bike
{
    your code here
     .............
     .............
}
```

Object It also considered as an "Instance of a Bike Class"

```
Bike objBike = new Bike();
```

Accessing Bike class methods

```
objBike.Color();
objBike.GetMileage();
```

OOP   Lab 1

File - New - Project – Console App

```
using System;

namespace Lab1
{
    class Program
    {
                                        Class Object

        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Create a Class

To create a class, use the `class` keyword:

Create a class named "`Car`" with a variable `color`:

```
class Car
{
  string color;    // string color = "red";

}
```

When a variable is declared directly in a class, it is often referred to as a **field** (or attribute).
It is not required, but it is a good practice to start with an uppercase first letter when naming classes.

```
namespace Example
{
class Vehicle
    {
        string name;
        string model;
        string color;
        int engineSize;

        public void starting()
        {}
        public void stopping()
        {}

    }

class program
    {
        static void main(string[] args)
        {
            Vehicle Toyota = new Vehicle();

        }
    }
}
```

Attributes

Methods

Class

Object

```
class Car
{
string model;
 string color;
 int speed;

 public void fullInfo()        // method
 {
   Console.WriteLine("The Model is: " + model);
   Console.WriteLine("The Color is: " + color);
   Console.WriteLine("The Speed is: " + speed);
 }
}
```

Attributes

Methods

```
static void Main(string[] args)
{
Car myCar = new Car();      //  create object
myCar.model=”TOYOTA”;
Console.readln(myCar.model);
myCar.color=”RED”;
Console.readln(myCar.color);
myCar.speed=220;
Console.readln(myCar.speed);
myCar.fullInfo();                 // Call the method
Console.ReadKey();           //  Freeze Screen
}
}
```

Create an Object

An object is created from a class. We have already created the class named Car, so now we can use this to create objects.
To create an object of Car, specify the class name, followed by the object name, and use the keyword new:
Create an object called "myObj" and use it to print the value of color:

```
class Car
{
 string color;

static void Main(string[] args)
 {
  Car myObj = new Car();

  Console.WriteLine(myObj.color);
 }
}
```

Note that we use the dot syntax (.) to access variables/fields inside a class (myObj.color).

Multiple Objects

You can create multiple objects of one class:

```
class Car
{
  string color = "red";
}

static void Main(string[] args)
{
  Car myObj1 = new Car();
  Car myObj2 = new Car();

  Console.WriteLine(myObj1.color);
  Console.WriteLine(myObj2.color);
}
}
```

Using Multiple Classes

You can also create an object of a class and access it in another class.
This is often used for better organization of classes (one class has all the fields and methods, while the other class holds the Main() method (code to be executed)).

```
class Program
{
   class Car
    {
      string color = "red";
    }

 static void Main(string[] args)
 {
  Car myObj = new Car();

  Console.WriteLine(myObj.color);
 }
}
```

Class Members

Fields and methods inside classes are often referred to as "Class Members":

```
// The class
class MyClass
{
// Class members
string color = "red";   // field
int maxSpeed = 200;     // field

public void fullThrottle() // method
{
Console.WriteLine("The car is going as fast as it can!");
}
}
```

Fields

variables inside a class are called fields, and that you can access them by creating an object of the class, and by using the dot syntax (.).

```
class Car
{
  string color = "red";
  int maxSpeed = 200;
}
static void Main(string[] args)
{
Car myObj = new Car();
Console.WriteLine(myObj.color);
Console.WriteLine(myObj.maxSpeed);
}
}
```

```
class Car
{
  string color;
  int maxSpeed;
  static void Main(string[] args)
  {
    Car myObj = new Car();
    myObj.color = "red";
    myObj.maxSpeed = 200;
    Console.WriteLine(myObj.color);
    Console.WriteLine(myObj.maxSpeed);
}
}
```

This is especially useful when creating multiple objects of one class:

```
class Car
{
  string model;
  string color;
  int year;
static void Main(string[] args)
{
   Car Ford = new Car();
   Ford.model = "Mustang";
   Ford.color = "red";
   Ford.year = 1969;
   Car Opel = new Car();
   Opel.model = "Astra";
   Opel.color = "white";
   Opel.year = 2005;
   Console.WriteLine(Ford.model);
   Console.WriteLine(Opel.model);
}
}
```

Object Methods

methods are used to perform certain actions.
Methods normally belongs to a class, and they define how an object of a class behaves.
Just like with fields, you can access methods with the dot syntax. However, note that the method must be public. And remember that we use the name of the method followed by two parantheses () and a semicolon ; to call (execute) the method:

```
class Car
{
  string color; // field
  int maxSpeed; // field
  public void fullThrottle() // method
  {
    Console.WriteLine("The car is going as fast as it can!");
  }
static void Main(string[] args)
{
  Car myObj = new Car();
  myObj.fullThrottle(); // Call the method
}
}
```

Use Multiple Classes

Remember from the last chapter, that we can use multiple classes for better organization (one for fields and methods, and another one for execution). This is recommended:

```
class Car
{
  public string model;
  public string color;
  public int year;
  public void fullThrottle()
  {
    Console.WriteLine("The car is going as fast as it can!");
  }
}
   class Program
   {
   static void Main(string[] args)
   {
     Car Ford = new Car();
     Ford.model = "Mustang";
     Ford.color = "red";
     Ford.year = 1969;
     Car Opel = new Car();
     Opel.model = "Astra";
     Opel.color = "white";
     Opel.year = 2005;
     Console.WriteLine(Ford.model);
     Console.WriteLine(Opel.model);
   }
   }
```

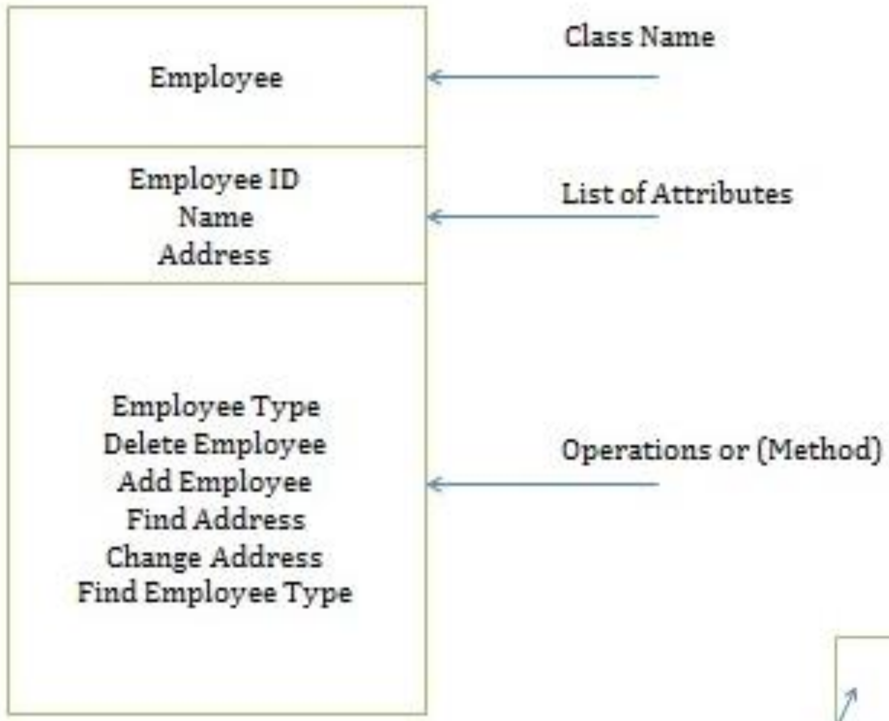
Unified Modeling Language (UML)

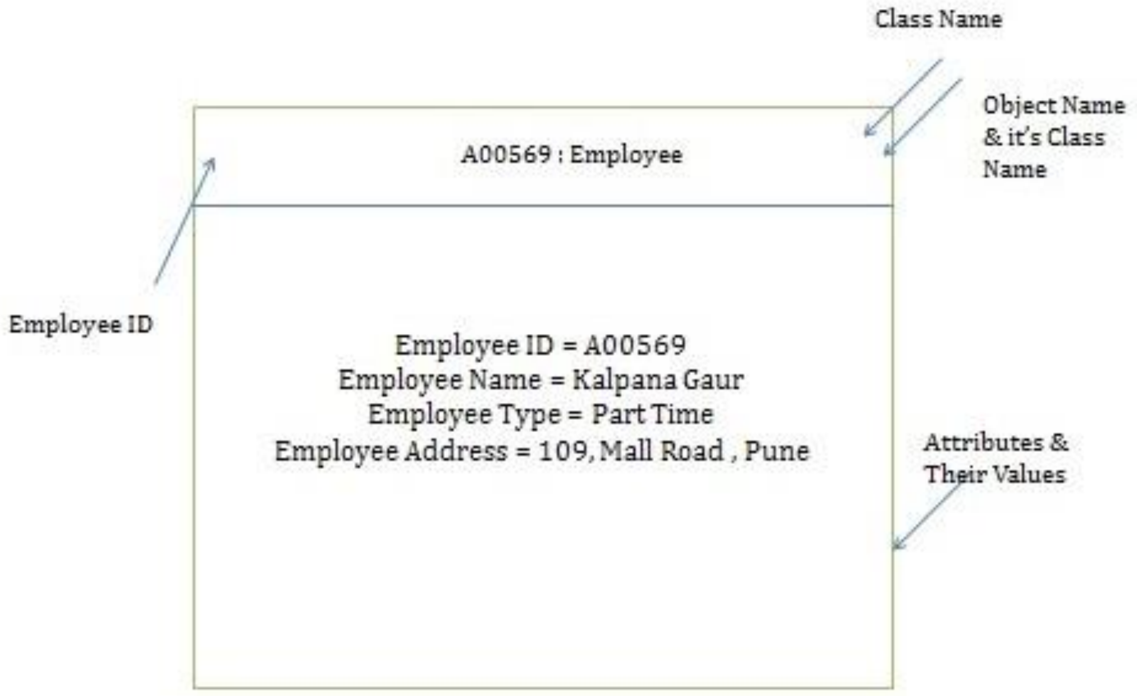
UML is a visual language that lets you to model processes, software, and systems to express the design of system architecture. It is a standard language for designing and documenting a system in an object oriented manner that allow technical architects to communicate with developer.
It is defined as set of specifications created and distributed by Object Management Group. UML is extensible and scalable.
The objective of UML is to provide a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation.
UML is made up of −
•**Diagrams** − It is a pictorial representations of process, system, or some part of it.
•**Notations** − It consists of elements that work together in a diagram such as connectors, symbols, notes, etc.
Example of UML Notation for class

UML

Unified Modeling Language,

is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. In this article, we will give you detailed ideas about what is UML, the history of UML and a description of each UML diagram type, along with UML examples.


Employee
Employee ID
Name
Address
Employee Type
Delete Employee
Add Employee
Find Address
Change Address
Find Employee Type
Class Name
List of Attributes
Operations or (Method)


Class Name
Object Name & it's Class Name
A00569 : Employee
Employee ID
Employee ID = A00569
Employee Name = Kalpana Gaur
Employee Type = Part Time
Employee Address = 109, Mall Road , Pune
Attributes & Their Values

Uses of UML

UML is quite useful for the following purposes −

•Modeling the business process

•Describing the system architecture

•Showing the application structure

•Capturing the system behavior

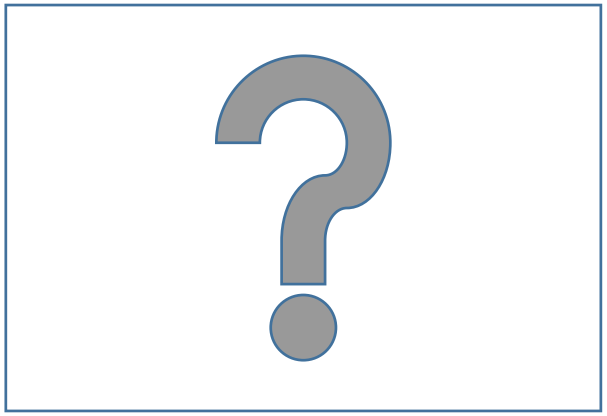
•Modeling the data structure

•Building the detailed specifications of the system

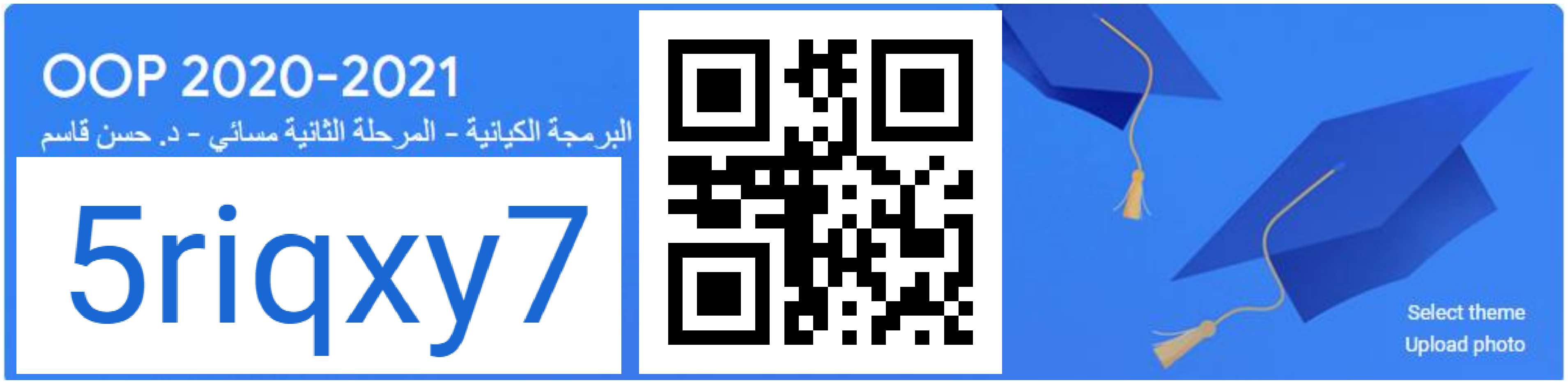
•Sketching the ideas

•Generating the program code

QUESTION



Google Classroom :



OOP 2020-2021

البرمجة الكيانية - المرحلة الثانية مسائي - د. حسن قاسم

5riqxy7