Object Oriented Programming

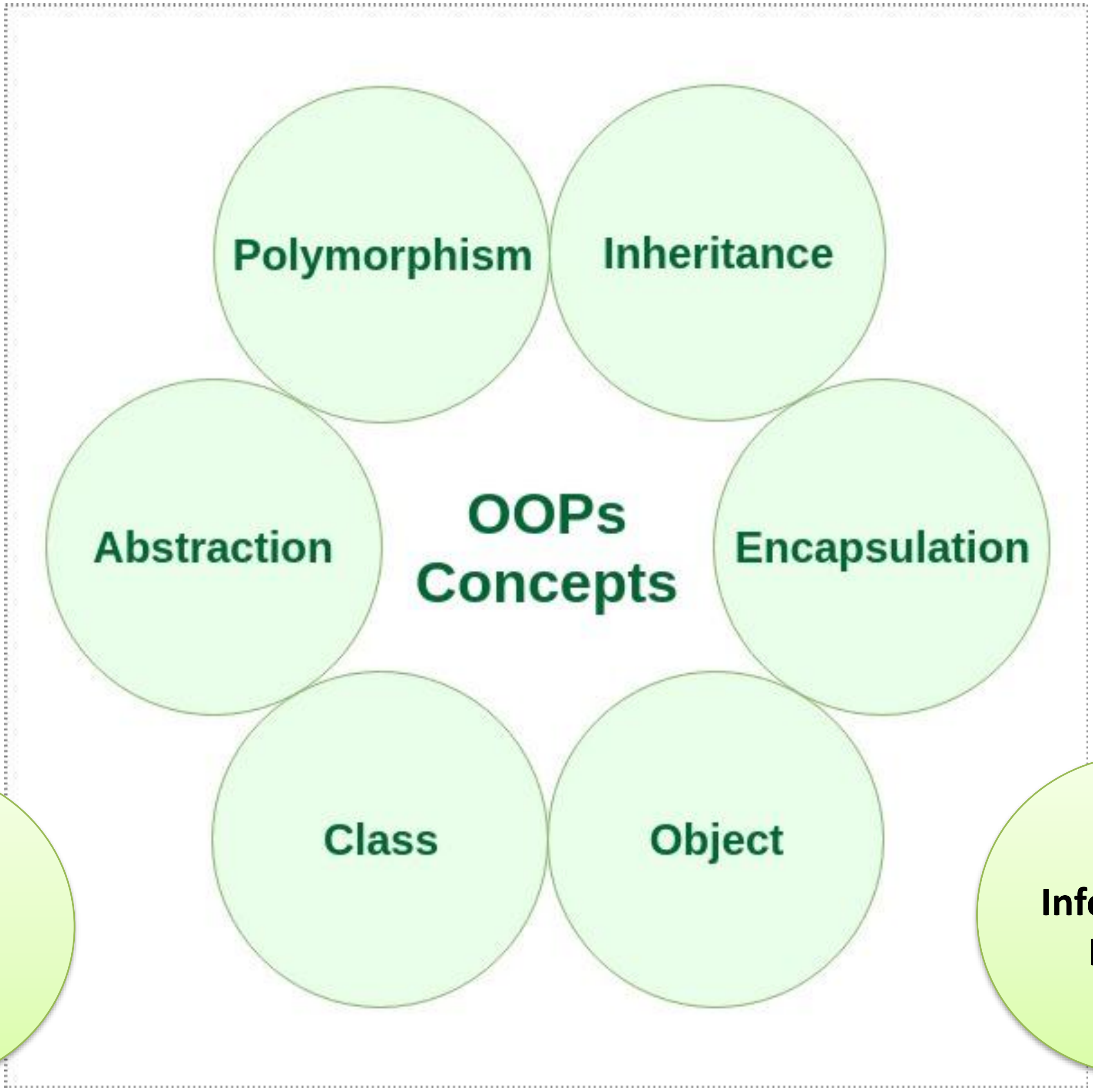
OOP

Object-Oriented Programming

Object-oriented programming is a **programming methodology** characterized by the following concepts:

1. **Data Abstraction**: problem solving via the formulation of abstract data types (ADT's).
2. **Encapsulation**: the proximity of data definitions and operation definitions.
3. Information hiding: the ability to selectively hide implementation details of a given ADT.
4. **Polymorphism**: the ability to manipulate different kinds of objects, with only one operation.
5. **Inheritance**: the ability of objects of one data type, to inherit operations and data from another data type.
6. **Class - Object**

Basic Concepts of OOP (SYSTEM)



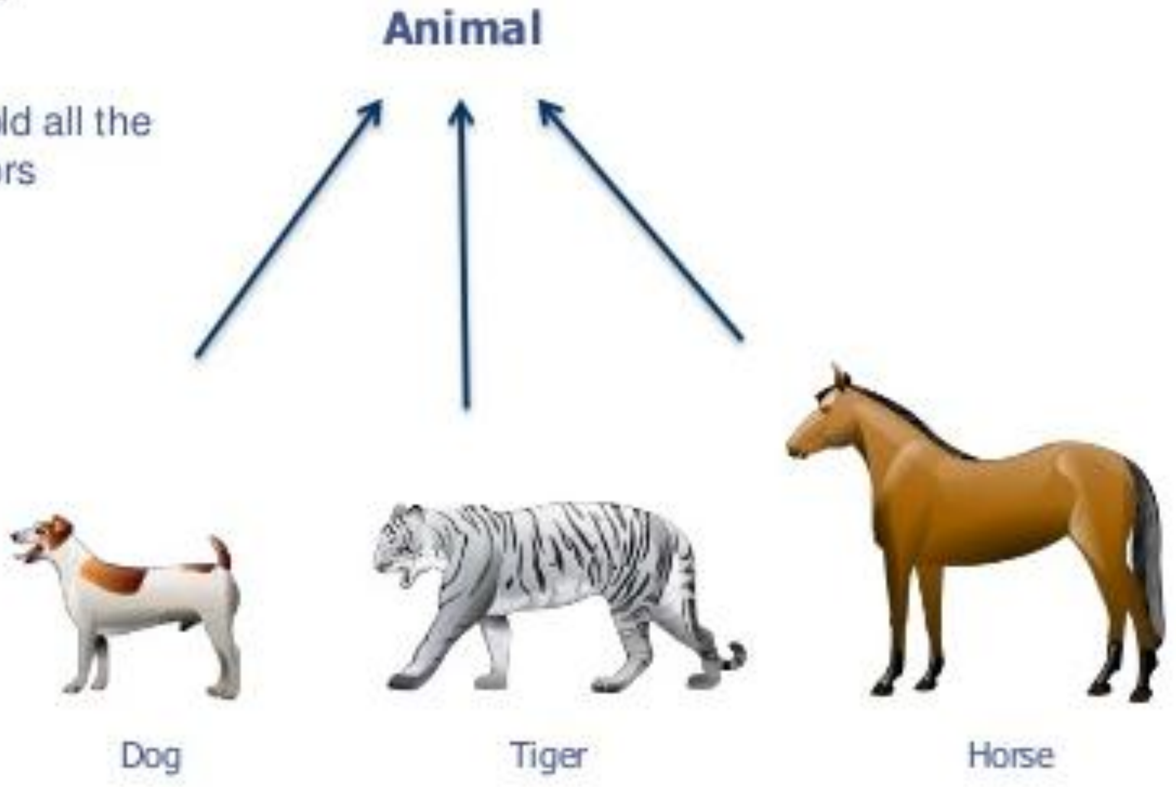
(dynamic binding)

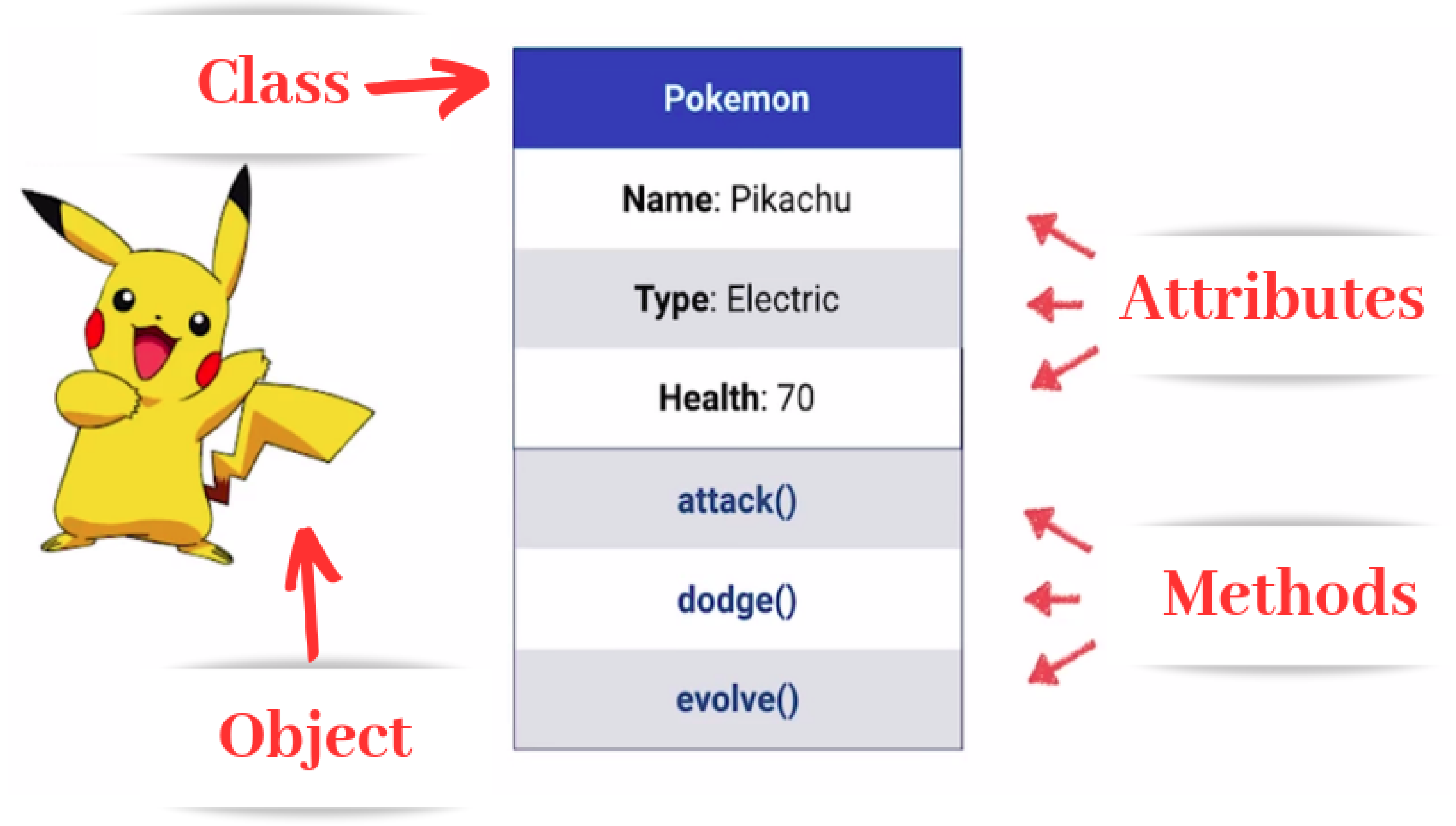
Information Hiding

Abstraction
Example

The wrapping up of data and function into a single unit (called class)

- **Animal** can be the abstraction for a dog, tiger, horse etc. It acts as a super-categorical noun.
- As an abstract concept it can hold all the common properties and behaviors present in different species



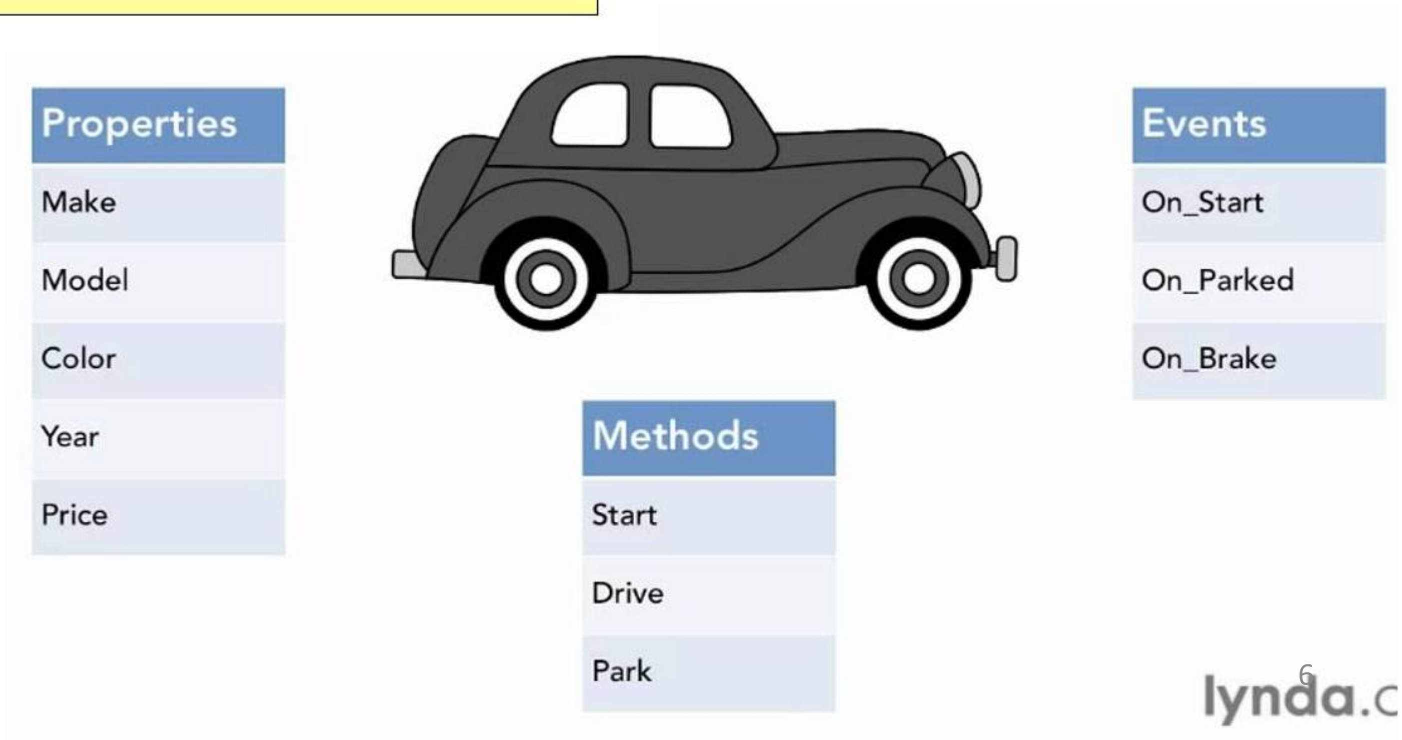


Class Pokemon : Pikachu

The attributes are some time called *data members* because they hold information. The functions that operate on these data are sometimes called *methods or member function*.

Characteristics/Attributes
- Make
- Model
- Number of doors
- Engine size

Actions it can perform/Methods
- Accelerate
- Stop
- Brake
- Turn



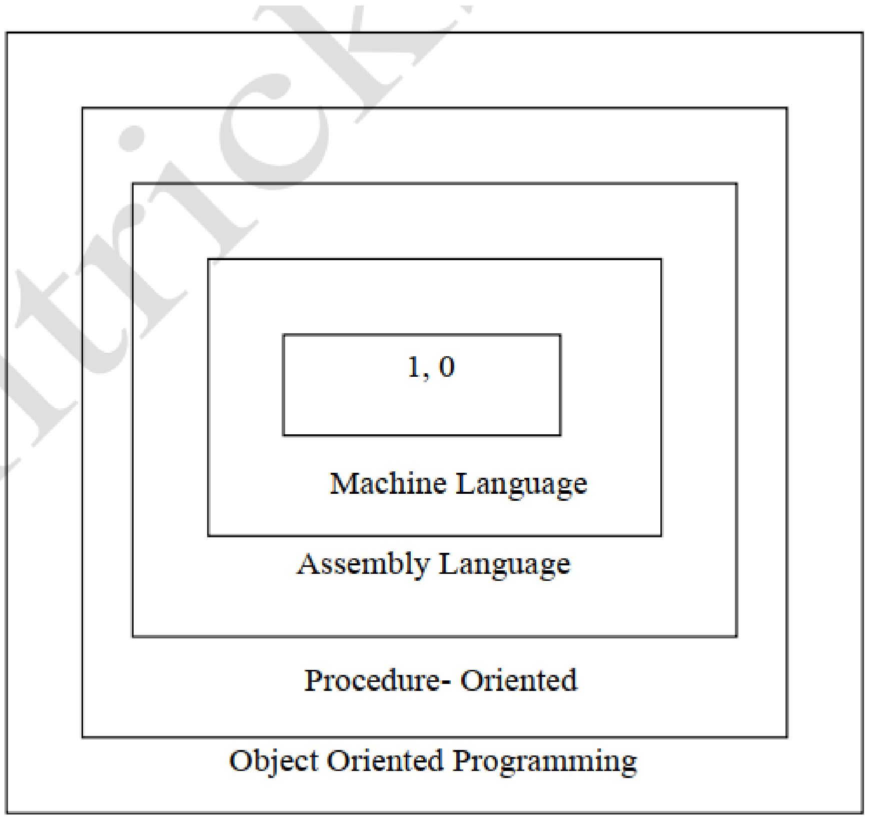
Properties
Make
Model
Color
Year
Price

Methods
Start
Drive
Park

Events
On_Start
On_Parked
On_Brake

Programming Languages

- Programming languages allow programmers to code software.
- The three major families of languages are:
  - Machine languages
  - Assembly languages
  - High-Level languages



Machine Languages

- Comprised of 1s and 0s
- The “native” language of a computer
- Difficult to program – one misplaced 1 or 0 will cause the program to fail.
- Example of code:

```
1110100010101        111010101110
10111010110100       10100011110111
```

Assembly Languages

- Assembly languages are a step towards easier programming.
- Assembly languages are comprised of a set of elemental commands which are tied to a specific processor.
- Assembly language code needs to be translated to machine language before the computer processes it.
- Example:

```
ADD  1001010, 1011010
```

High-Level Languages

- High-level languages represent a giant leap towards easier programming.
- The syntax of HL languages is similar to English.
- Historically, we divide HL languages into two groups:
  - Procedural languages
  - Object-Oriented languages (OOP)

Procedural Languages

- Early high-level languages are typically called procedural languages.
- Procedural languages are characterized by sequential sets of linear commands. The focus of such languages is on *structure*.
- Examples include C, COBOL, Fortran, LISP, Perl, HTML, VBScript

- **Examples of OOP languages include:**

C++, Visual Basic.NET and Java.

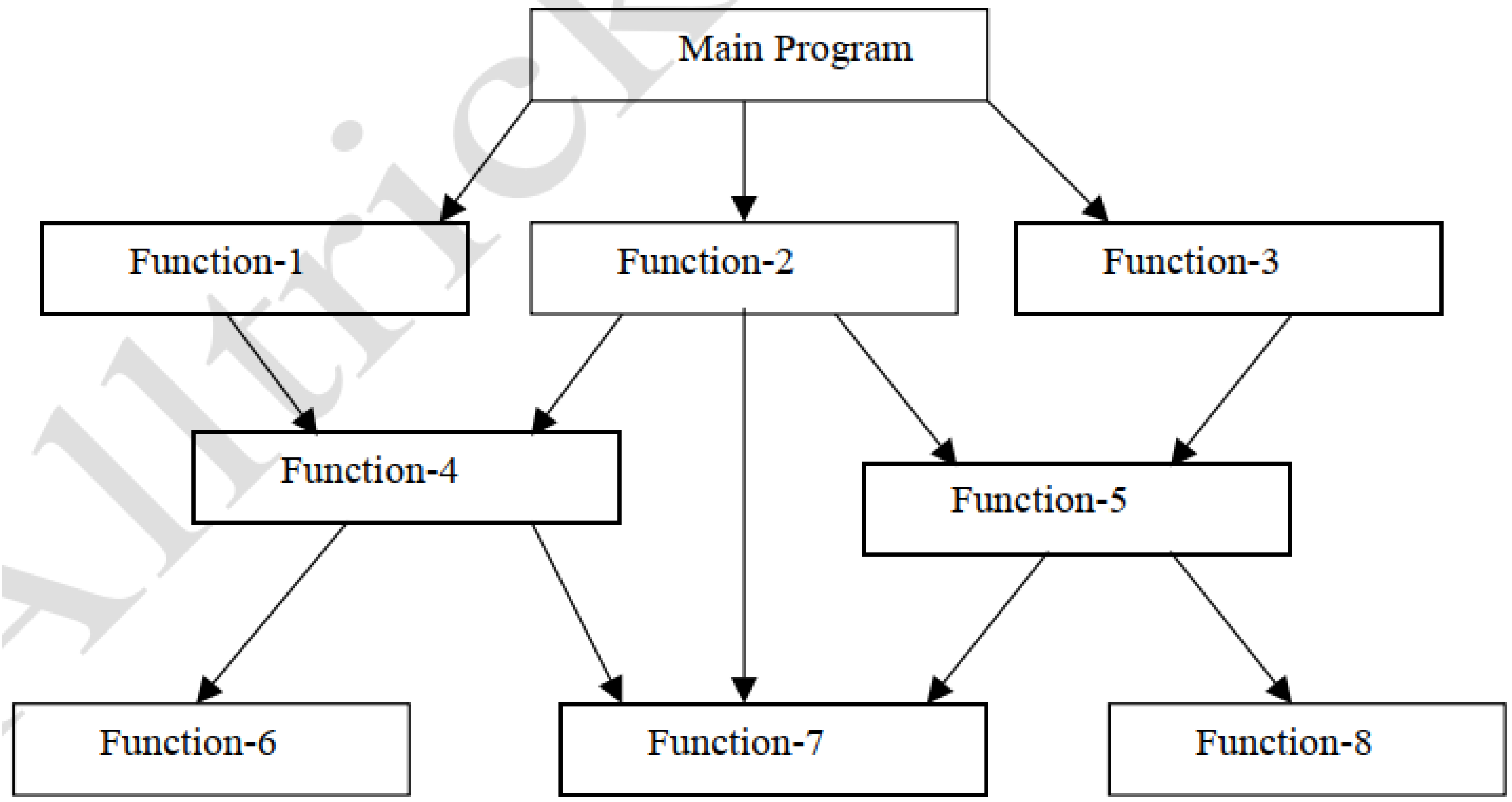


Fig. 1.2 Typical structure of procedural oriented programs

OOP Benefits

- Through inheritance, we can eliminate redundant code extend the use of existing Classes.
- The principle of data hiding helps the programmer to build secure program that can not be invaded by code in other parts of a programs.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is easy to partition the work in a project based on objects.
- Object-oriented system can be easily upgraded from small to large system.
- Message passing techniques for communication between objects makes to interface descriptions with external systems much simpler.
- Reusability

- **Some of the features of object oriented programming are:**

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are ties together in the data structure.
- Data is hidden and cannot be accessed by external function.
- Objects may communicate with each other through function.
- New data and functions can be easily added whenever necessary.
- Follows bottom up approach in program design.

Application of OOP

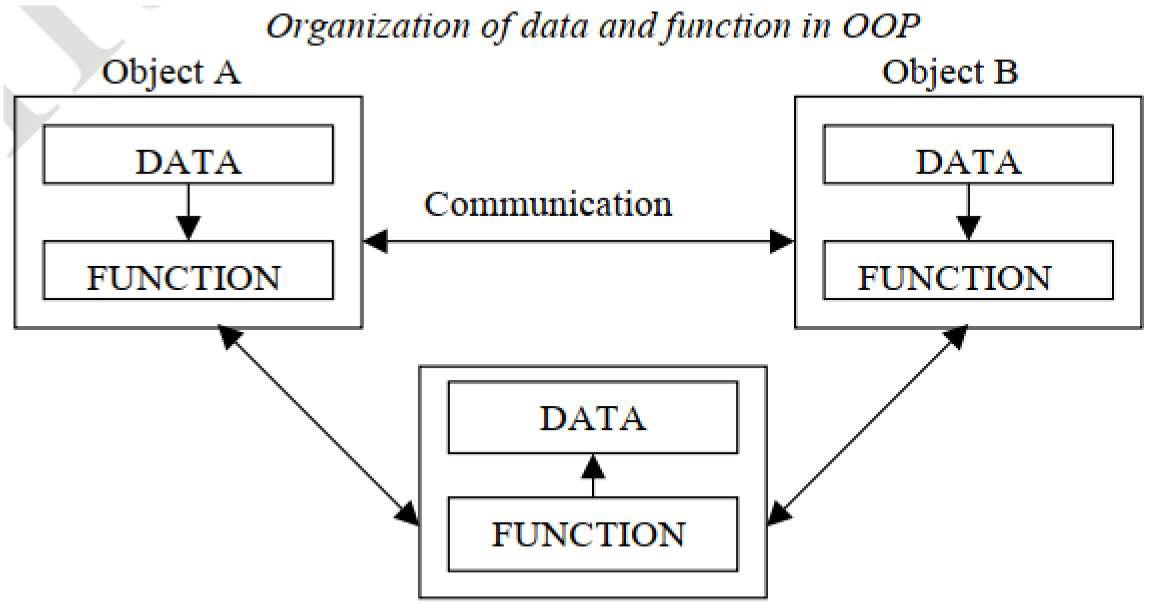
- Real-time system
- Simulation and modeling
- Object-oriented data bases
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CAM/CAD systems

OOP environment will enable the software industry to improve not only the quality of software system but also its productivity. Object-oriented technology is certainly going to change the way the software engineers think, analyze, design and implement future system.

Object Oriented Programming

- **Object** – Unique programming entity that has *methods, has attributes* and can react to *events*.

- **Attribute** – Things which describe an object; the "adjectives" of objects. In code, usually can be identified by a "descriptive" word – *Enabled, BackColor*
- **Method** – Things which an object can do; the "verbs" of objects. In code, usually can be identified by an "action" like *Hide, Show*

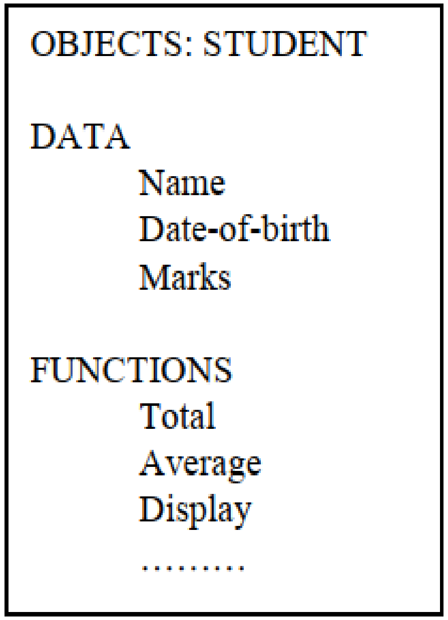


Object Oriented Programming

- **Class** – Provides a way to create new objects based on a “meta-definition” of an object (Example: The automobile **class**)
- **Constructors** – Special methods used to create new instances of a class (Example: A Honda Civic is an **instance** of the automobile **class**.)

Classes and Objects

- 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).
- An **object** is an *instance* of a class. With one class, you can have as many objects as required.



OBJECTS: STUDENT

DATA
- Name
- Date-of-birth
- Marks

FUNCTIONS
- Total
- Average
- Display
- ....

Fig. 1.5 representing an object

- **Objects** are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.

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"*

Abstraction

- Abstraction is a design principle.
- Is the process of removing characteristics from something in order to reduce it to a set of essential characteristics.
- Through the process of abstraction, a programmer hides all but the relevant data about a class in order to reduce complexity and increase reusability.

Abstraction

- Abstraction allows programmers to represent complex real world in the simplest manner.
- It is a process of identifying the relevant qualities and behaviors an object should possess, in other word represent the necessary features without representing the back ground details
- You should always use abstraction to ease reusability, and understanding for the design and enable extension.
- When we design the abstract classes, we define the *framework* for later extensions.

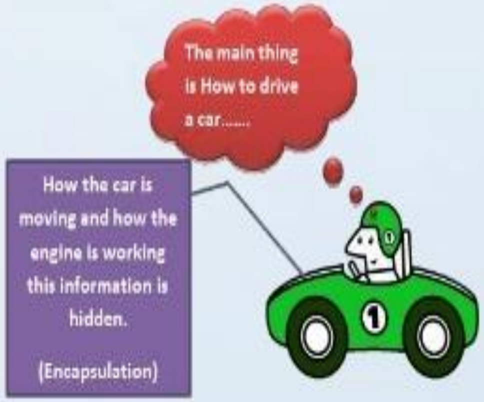
Abstraction

Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight, and cost,
and function operate on these attributes.
They encapsulate all the essential properties of the object that are to be created.

The **attributes** are some time called ***data members*** because they hold information. The **functions** that operate on these data are sometimes called ***methods or member function***.

Abstraction

- Abstraction is the "process of representing only essential features". That means Abstraction doesn't show the **complexity behind features**. Abstraction is used for "Making things more general, simple".
- Example : ATM Machine , Car

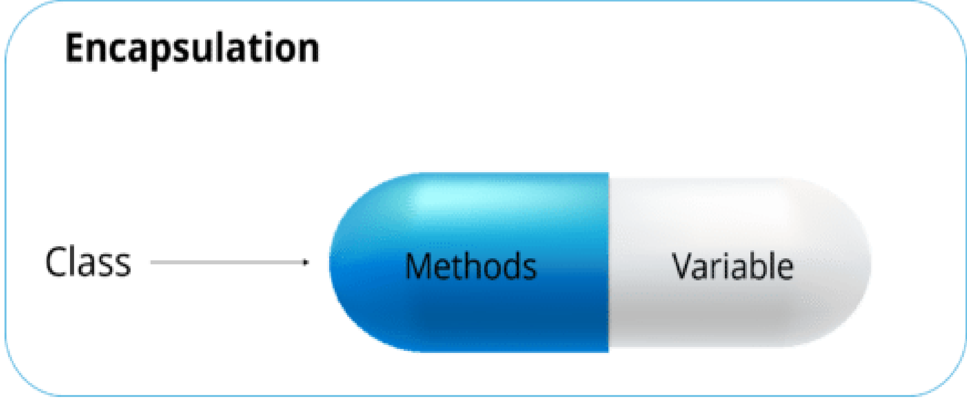

The main thing is How to drive a car......
How the car is moving and how the engine is working this information is hidden.
(Encapsulation)

Real Life Example of Abstraction

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..

Encapsulation



- Incorporation into a class of data & operations in one package
- Data can only be accessed through that package
- “Information Hiding”

The wrapping up of data and function into a single unit (called class).
The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object’s data and the program.
This insulation of the data from direct access by the program is called *data hiding or information hiding*

Encapsulation

- Is the inclusion of property & method within a class/object in which it needs to function properly.
- Also, enables reusability of an *instant* of an already implemented class within a new class while ***hiding & protecting*** the method and properties from the client classes.

Encapsulation – Benefits

- Ensures that structural changes remain local:
  - Changing the class internals does not affect any code outside of the class
  - Changing methods' implementation does not reflect the clients using them
- Encapsulation allows adding some logic when accessing client's data
  - E.g. validation on modifying a property value
- Hiding implementation details reduces complexity → easier maintenance

Polymorphism

- Creating methods which describe the way to do some general function (Example: The “drive” method in the automobile class)
- Polymorphic methods can adapt to specific types of objects.

Polymorphism, a Greek term, means the ability to take more than on form.
An operation may exhibit different behavior is different instances. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.
The process of making an operator to exhibit different behaviors in different instances is known as operator overloading.

Polymorphism

- Polymorphisms is a generic term that means 'many shapes'. More precisely *Polymorphisms* means the ability to request that the same *methods* be performed by a wide range of different types of things.
- In *OOP*, *polymorphisms* is a technical issue and principle.
- It is achieved by using many different techniques named method overloading, operator overloading, and method overriding.

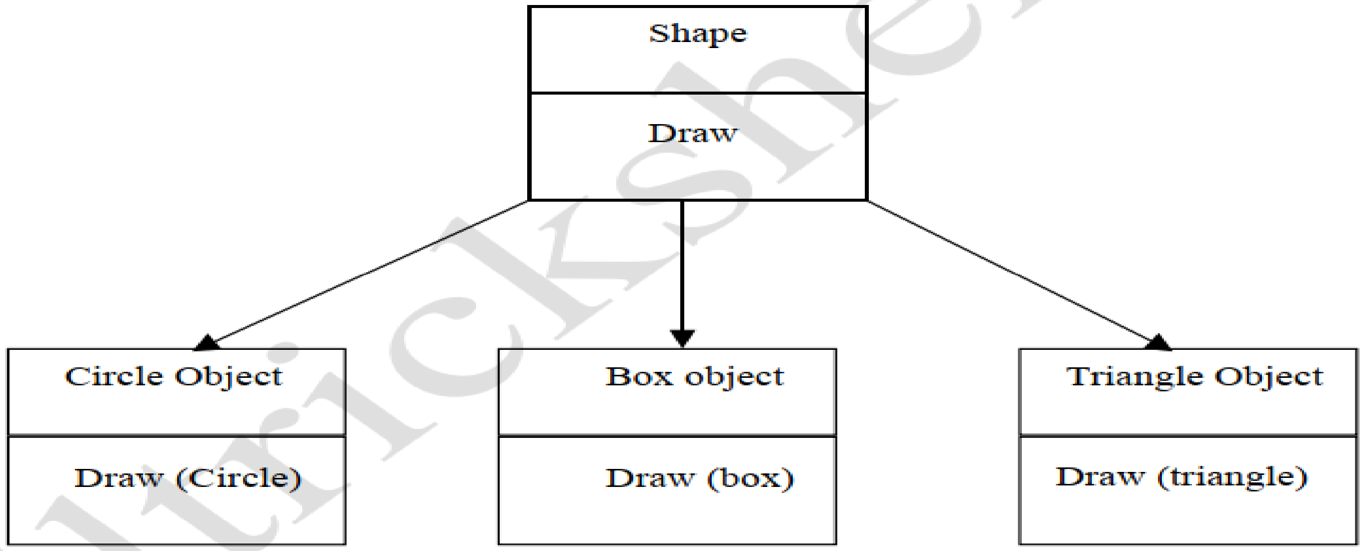


Fig. 1.7 Polymorphism

- **Dynamic Binding**
- Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. It is associated with polymorphism and inheritance. A function call associated with a polymorphic reference depends on the dynamic type of that reference.

Inheritance

Inheritance is the process by which objects of one class acquired the properties of objects of another classes.

- Allows programmers to create new classes based on an existing class
- Methods and attributes from the parent class are inherited by the newly-created class
- New methods and attributes can be created in the new class, but don’t affect the parent class’s definition

- It supports the concept of hierarchical classification.

For example, the bird, ‘robin’ is a part of class ‘flying bird’ which is again a part of the class ‘bird’.
The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived

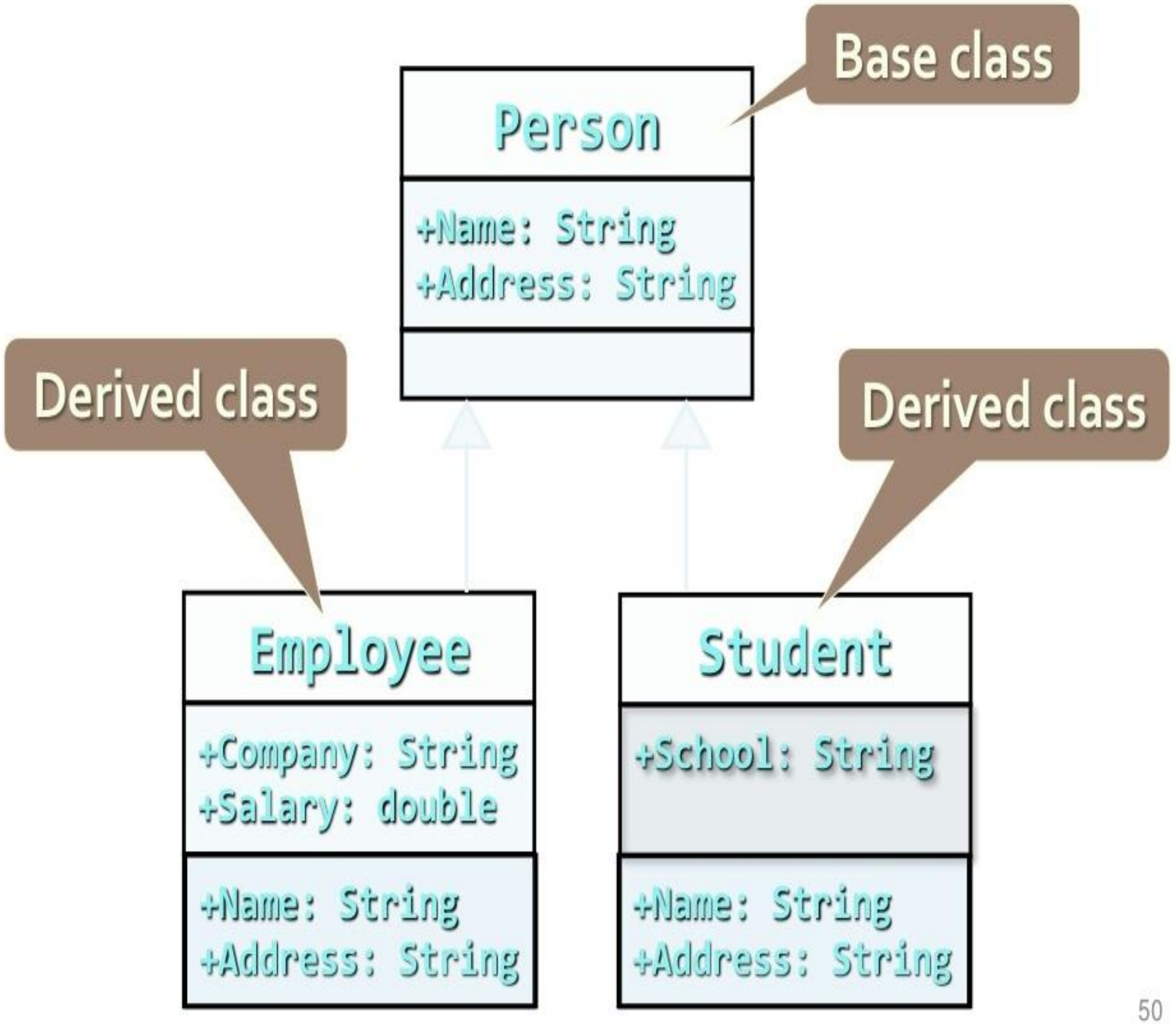
Inheritance

- Inheritance is defined as "one class (child class) inherits or acquire the property (members) of another Base or head class"
- Example : A son have all the properties of his grandfather and father but he have his own unique properties also.

Inheritance

- Inheritance allows child classes to inherit the characteristics of existing parent class
  - Attributes (fields and properties)
  - Operations (methods)
- Child class can extend the parent class
  - Add new fields and methods
  - Redefine methods (modify existing behavior)
- A class can implement an interface by providing implementation for all its methods

Inheritance – Example

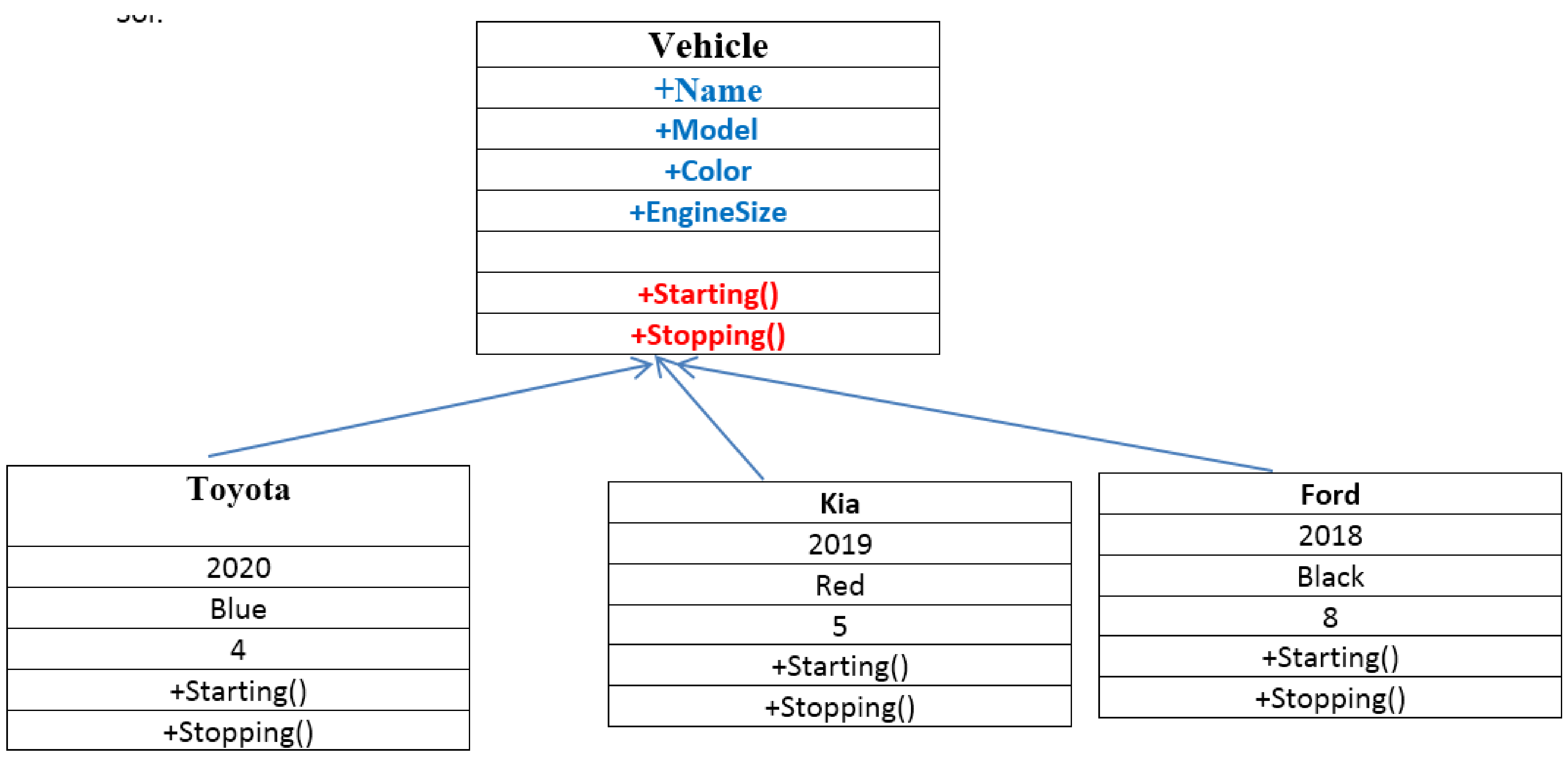


Write a C# program which creates a class for vehicle company with information below?.

Vehicle Name	Toyota	Kia	Ford
Model	2020	2019	2018
Color	Blue	Red	Black
EngineSize	4	5	8

Note: 1- All Vehicle can do Starting and Stopping. 2-Represent table information in UML form?

UML representation



```
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();

        }
    }
}
```

Class

Object