SmartCliff
CAREER MOBILITY SOLUTIONS

SDE Readiness Training

Empowering Tomorrow's Innovators

Module I

Java Software Development: Effective Problem Solving

SmartCliff
Career Mobility Solutions

OOPs Concept Realization

Learning Level: Basic

DATE: 02-07-2025

CONTENTS

Programming Paradigm

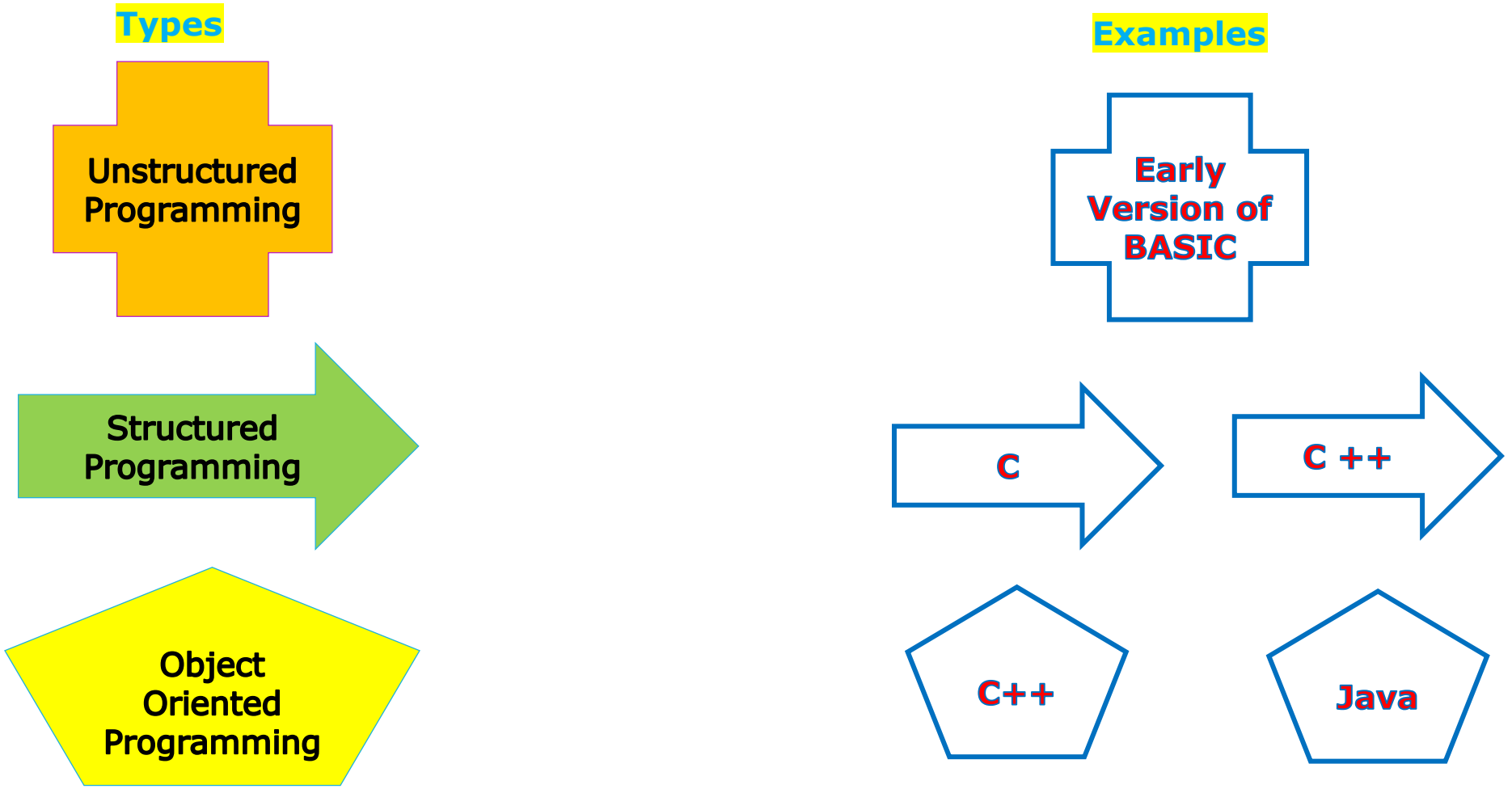
Overview

- Programming paradigm is a **way or style** of programming.
- It defines the **methodology of designing** and **implementing programs**.

Three basic types

1. **Unstructured Programming**: It contains only **one section, i.e., main**. The main consists of a sequence of commands and statements i.e. unable to use **selection and repetition control statement** constructs.
2. **Structured Programming**: It uses the structured control flow constructs of **selection** (if/then/else) and **repetition** (while and for), block structures, and **subroutines**.
3. **Object-Oriented Programming:** Organized **around objects.** It combines **data** and **action**.

Overview


Types
Unstructured Programming
Structured Programming
Object Oriented Programming
Examples
Early Version of BASIC
C
C ++
C++
Java

Unstructured Programming

- **Unstructured programming** is the historically earliest programming paradigm capable of creating Turing-complete algorithms.
- It uses unstructured **jumps** to labels or instruction addresses with the use of unstructured control flow using **goto** statements or equivalent.
- Unstructured programming has been heavily criticized for producing **hardly-readable** ("spaghetti") code.

Unstructured Programming

Advantages

- Simple and easy to practice by novice programmer
- Good for small program; no lengthy planning needed.
- Speed and flexible
- Provides full freedom to programmers to program as they want.

Disadvantages

- No code reusability
- Repetition of code
- Cumbersome Maintenance
- Debugging is tough

Structured Programming

- This programming paradigm aimed at **improving the clarity, quality, and development time** of a computer program by making extensive use of the structured control flow constructs of selection (if / then / else) and repetition (while and for), block structures, and **subroutines**.
- Structured programming enforces a **logical structure** on the program being written to make it more efficient and easier to understand and modify.
- The problem is **bifurcated into number of small problems** known as procedures (Also alternatively referred as functions or methods).

Structured Programming

Advantages

- Complexity can be reduced using modularity (divide and conquer)
- Logical structures ensure clear flow of control.
- Modules can be re-used many times; thus, it saves time and increase reliability.
- Easier to update/fix the program by replacing individual modules rather than larger amount of code.

Disadvantages

- Data is not secure
- Code reusability is less
- Can support the software development projects easily up to a certain level of complexity. If complexity of the project goes beyond a limit, it becomes difficult to manage.

Object Oriented Programming

Need

Programmers need a programming model that enables them to work with real-world entities / objects.

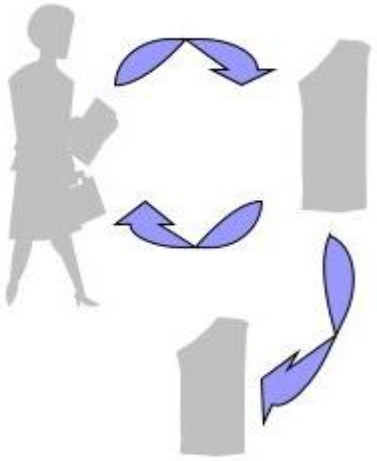
- People in real life have knowledge and can do a variety of tasks.
- Objects in OOP include fields to store **knowledge/state/data and methods** to do various tasks.

Structured Vs Object Oriented - Overview

Break down a programming task into a collection of **variables, data structures, and subroutines**.

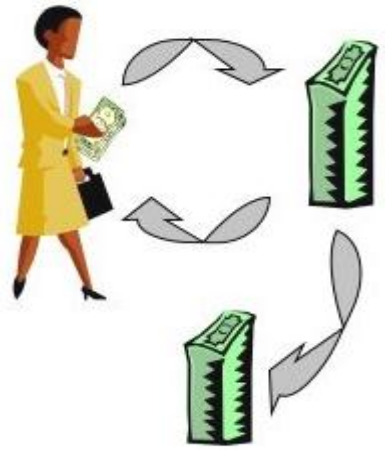
Break down a programming task into **objects** that expose behaviour (methods) and data (members or attributes).

Structured



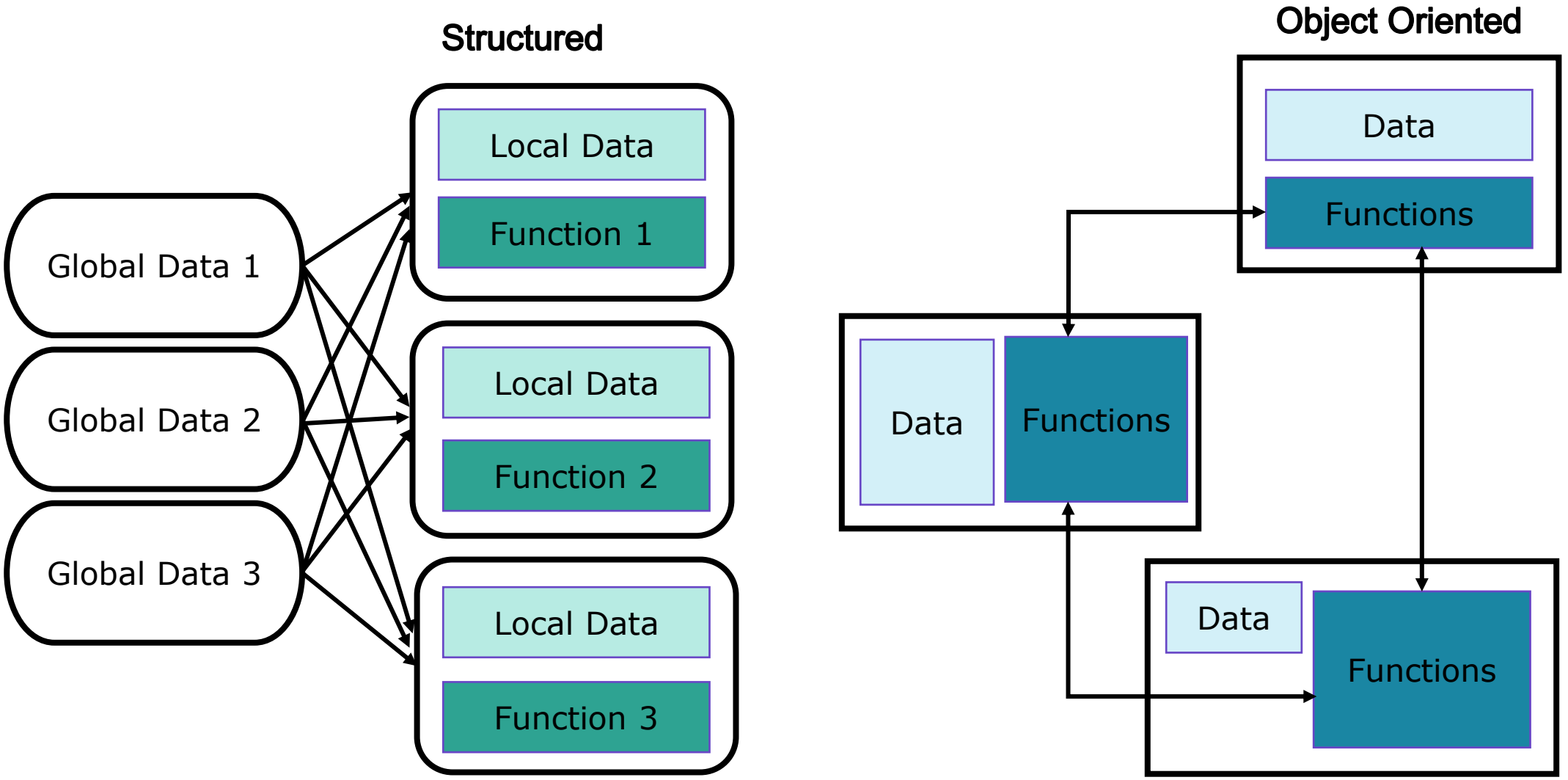
Subroutines:
Withdraw, Deposit, Transfer

Object Oriented

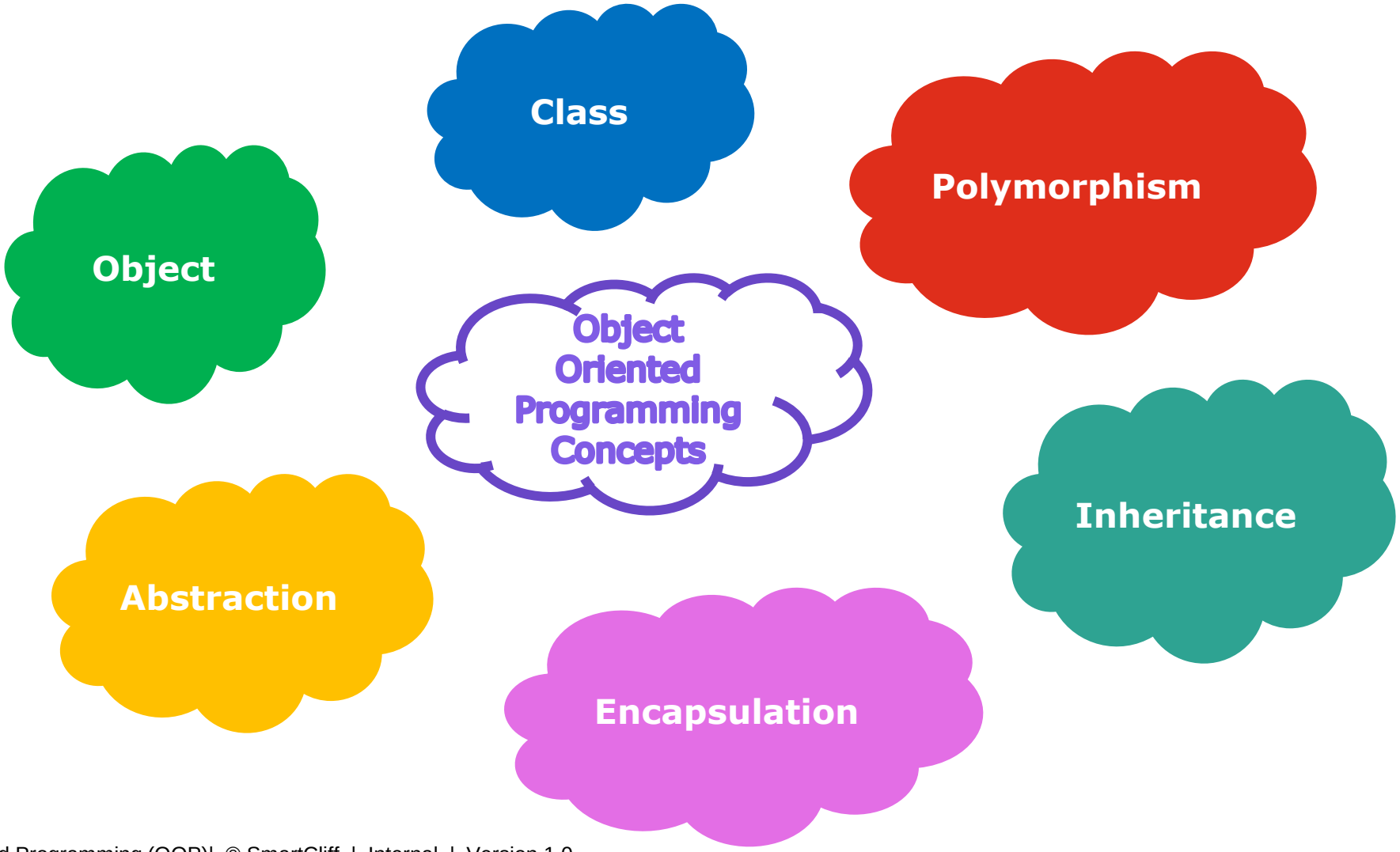


Objects:
Customer, Money, Account

Structured Vs Object Oriented - Overview



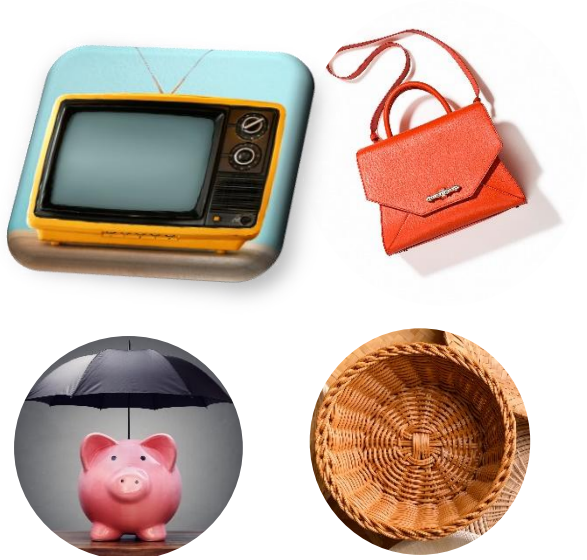
Features


Class
Polymorphism
Object
Object Oriented Programming Concepts
Inheritance
Abstraction
Encapsulation

Object - What it is?

- Object is the **key element** to understand *object-oriented* technology.
- It is a **real-world entity**.

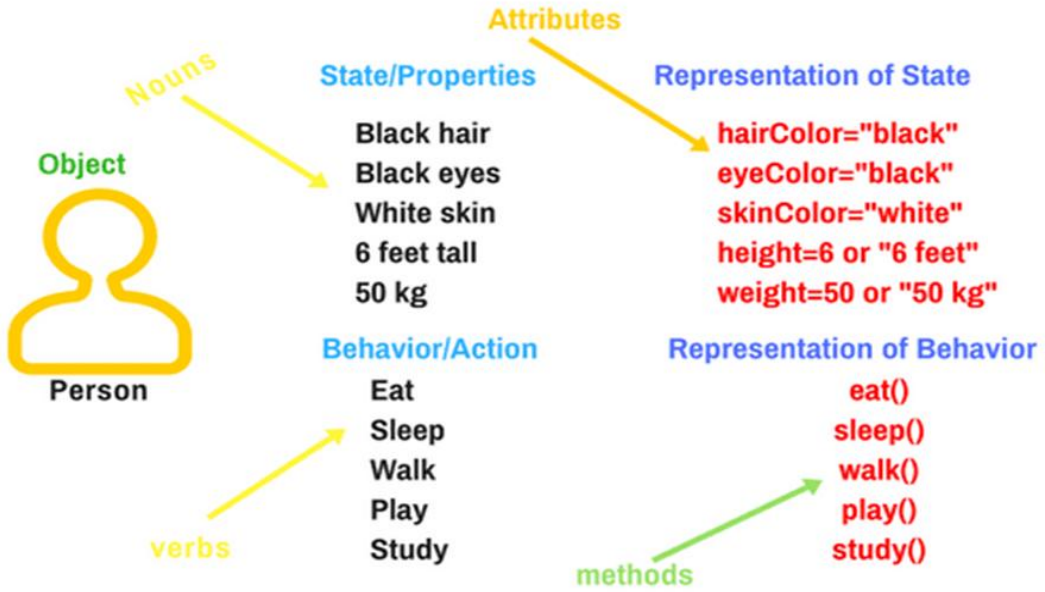
Real world Objects



Everything is an object in the real world

Object - What will it have?

- Real-world objects share **three characteristics**: State, Behavior and Identity.
  - The **state** of an object can be described as a set of attributes and their values.
  - The **behaviour /operation/action** of an object refers to the changes that occur in its attributes over a period.
  - **Identity** is the name that identifies the object.



Object – Real Life Example

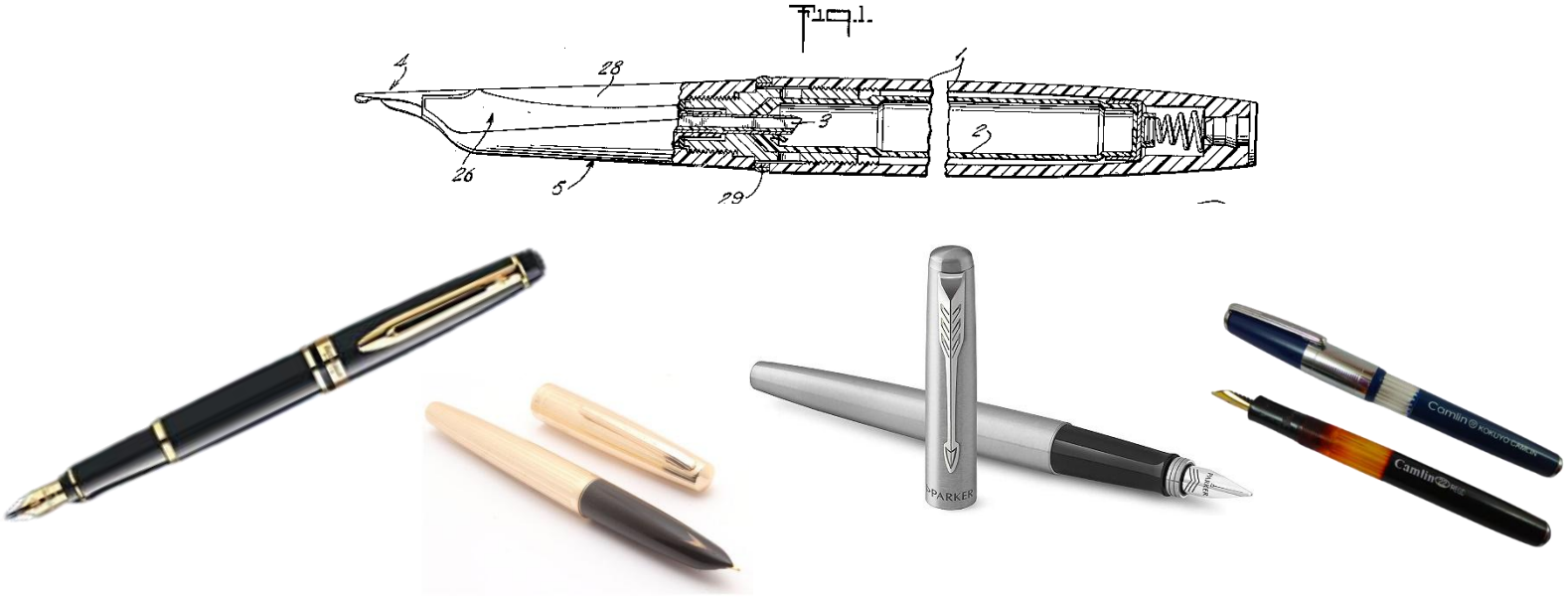
Object: Car

State / Properties	Representation of State
Silver body	color="silver"
12 lakh rupees	price="12 lakh rupees"
45000 kms	mileage="45000 kms"
Manual gear	transmission="manual gear"
Diesel engine	engine_type="diesel engine"
Behaviour / Action	**Representation of Behaviour**
Acceleration	acceleration()
Gear Change	gear_change()
Slowing Down	slowing_down()
Reversing	reversing()

Identity: It is a Hyundai Verna

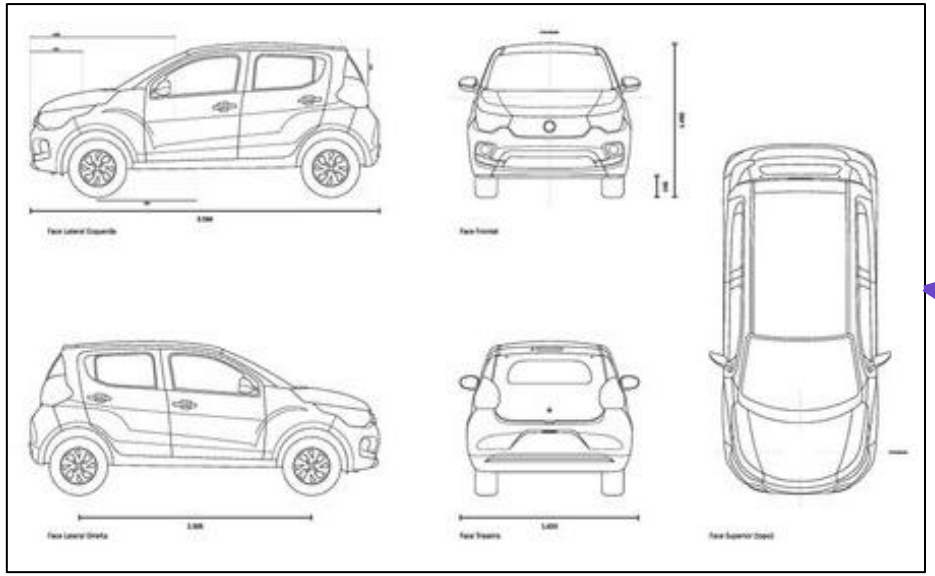
Class

- Class is a **group of objects with similar properties**, common **behaviour**, common **relationships** with other objects, and common **semantics**.
- A class is the **template / blueprint** from which individual objects are created.



Class – Real Life Example

Class Car



A class is like a blueprint for creating cars. It defines what features and behaviors a car can have. E.g., class specifies that every car should have attributes like color, brand, wheels, and behaviors like start, accelerate, stop. It is a group of individual cars with similar properties and actions.

Object and Class

Class – Car

States

- Color
- Transmission
- Brand
- Mileage
- Fuel Type

Behaviors

- Accelerate
- Reverse
- Park

Objects

Color: Silver
Transmission: Manual
Brand: Hyundai
Mileage: 45, 000 kms
Fuel Type: Diesel

A silver Hyundai car with manual gear, diesel fuel type, and 45,000 kms mileage.

Color: Red
Transmission: Automatic
Brand: Audi
Mileage: 47, 500 kms
Fuel Type: Electric

A red Audi with automatic gear, 47,500 kms mileage, and electric fuel type.

Color: Orange
Transmission: Manual
Brand: Tata
Mileage: 35, 800 kms
Fuel Type: Petrol

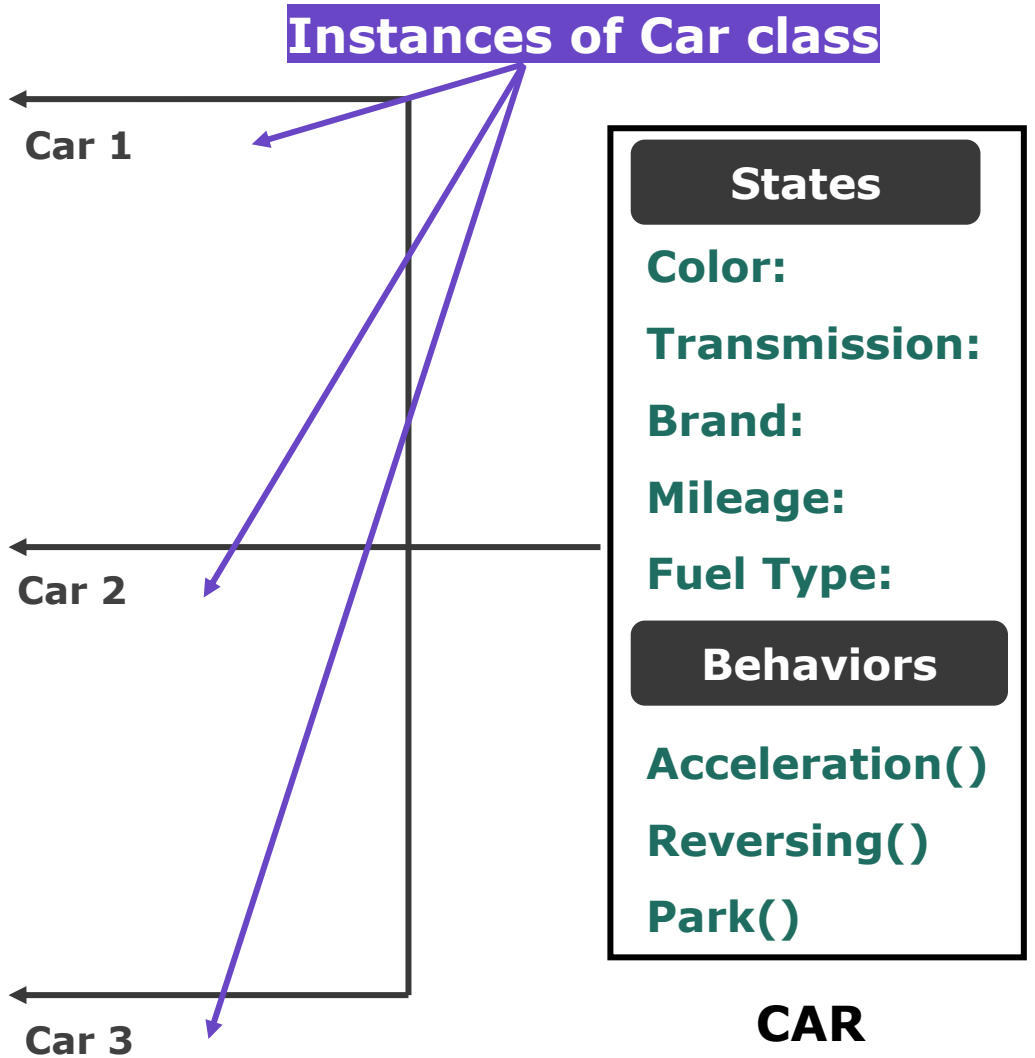
An orange Tata car with manual transmission, petrol fuel type, and 35,800 kms mileage.

Object – Instance of a class

Color: Silver
Transmission: Manual
Brand: Hyundai
Mileage: 45, 000 kms
Fuel Type: Diesel

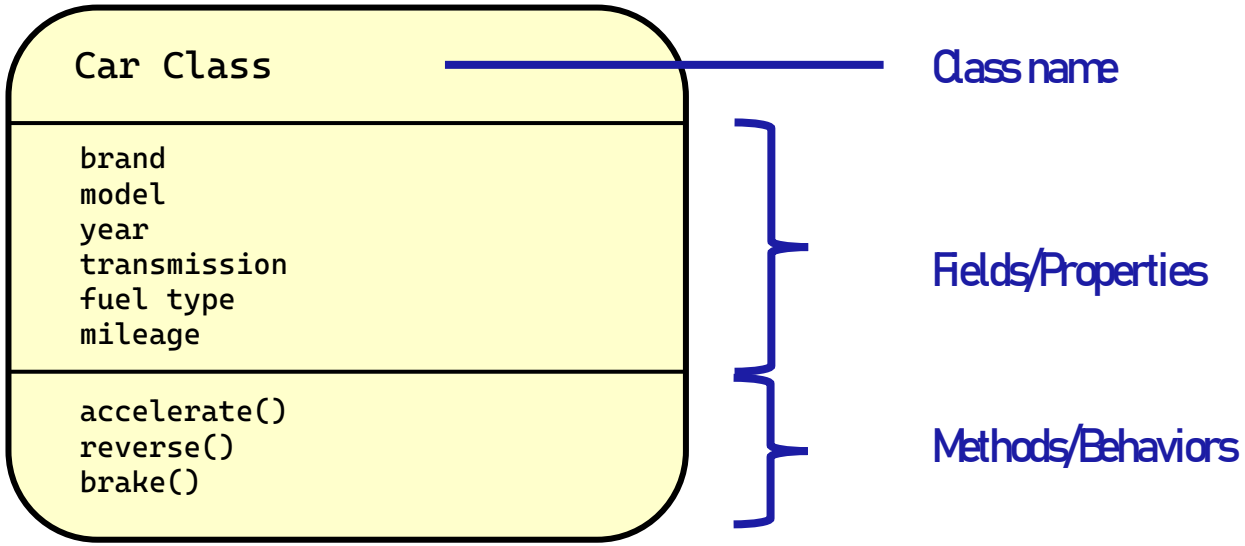
Color: Red
Transmission: Automatic
Brand: Audi
Mileage: 47, 500 kms
Fuel Type: Electric

Color: Orange
Transmission: Manual
Brand: Tata
Mileage: 35, 800 kms
Fuel Type: Petrol



Class diagram

Properties and Behaviors of 'Car' class



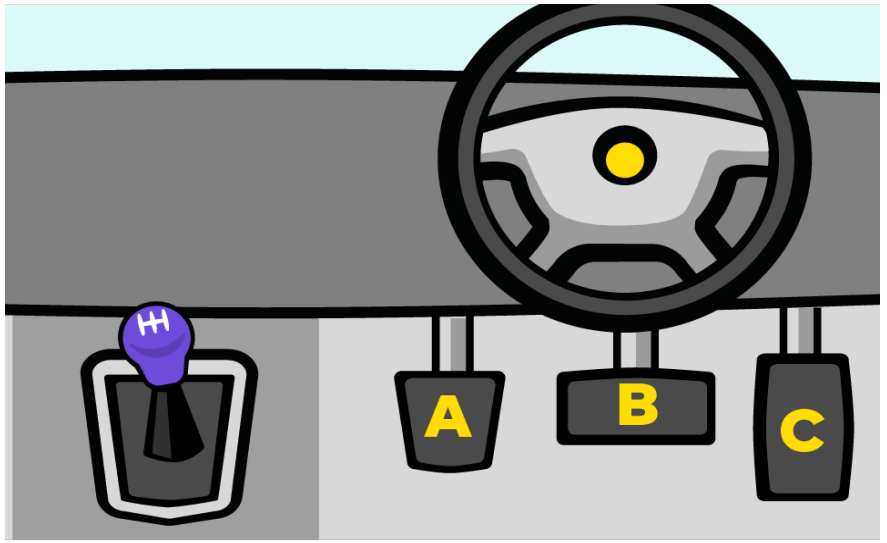
Abstraction

- Data abstraction is **providing** only what is **relevant/essential** information about the data to the outside world, **hiding** the **irrelevant/background details/implementation**.
  - **For Example,** when you **send an email** to someone you just click send and you get the success message.
  - **What actually happens when you click send?**
  - How data is transmitted over network to the recipient?
  - All these are hidden from you (**because it is irrelevant to you**).

Abstraction

ATM user need not to know what is the process behind the screen.

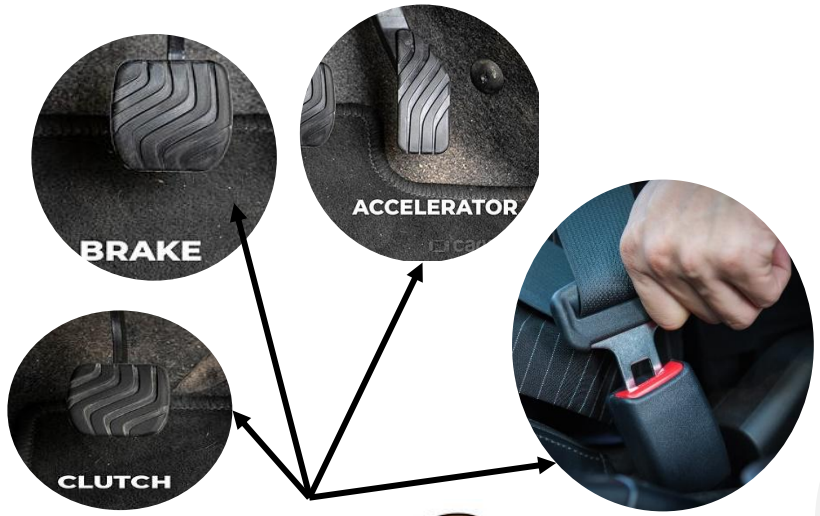
Musician need not to know how the sound is produced. All they need to know is which key to press

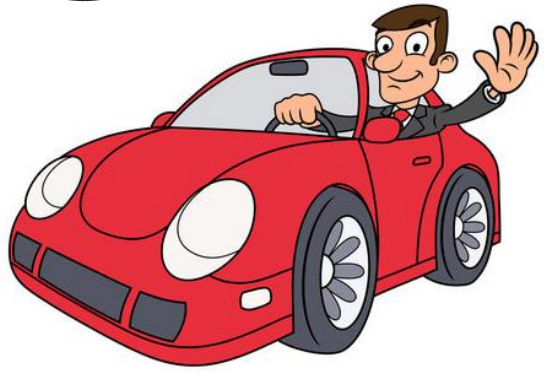


Car driver need not know how the engine works, all they need to know is which pedals to press and which gear to shift

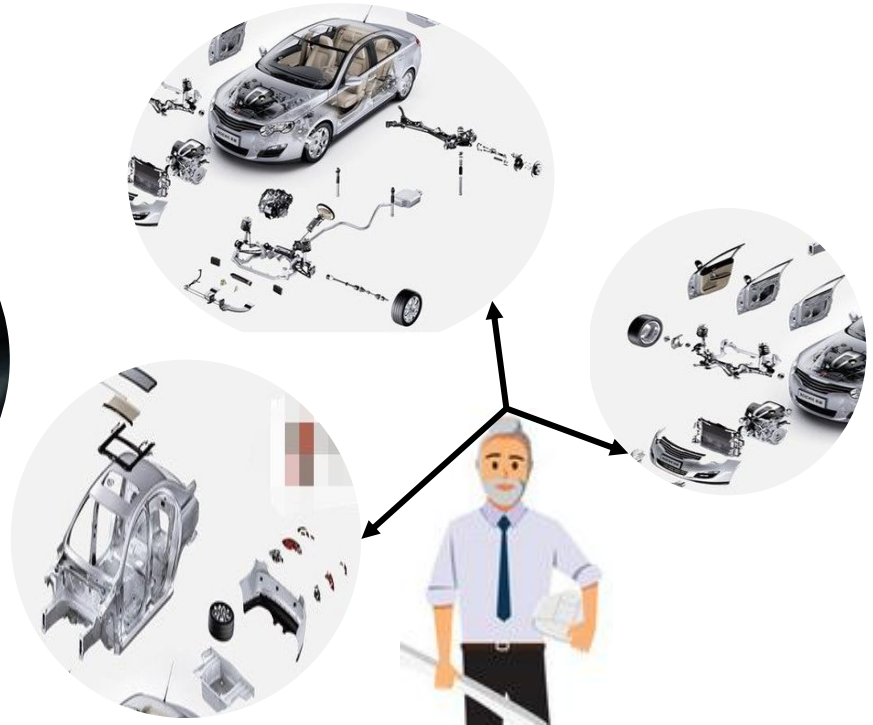
Note: Abstraction is providing only relevant/essential information to the user.

Abstraction – Real Life Example

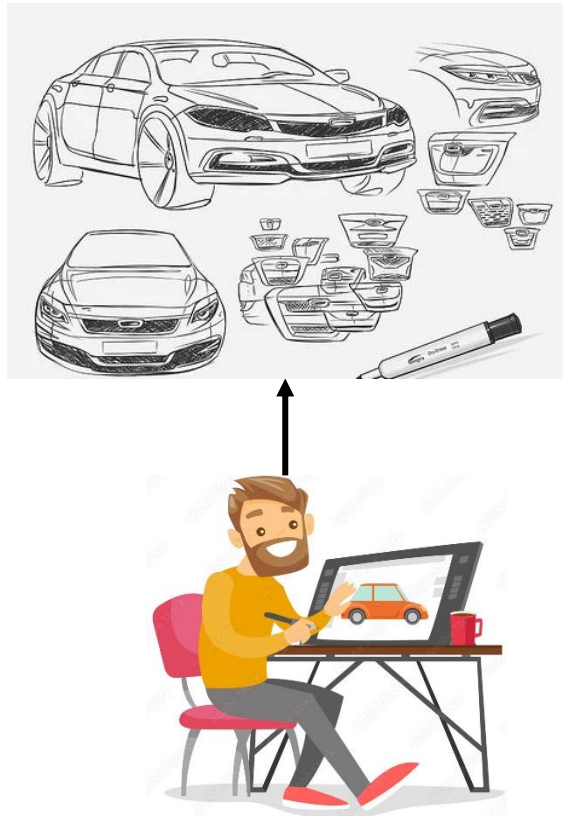




Driver / User



Engineers



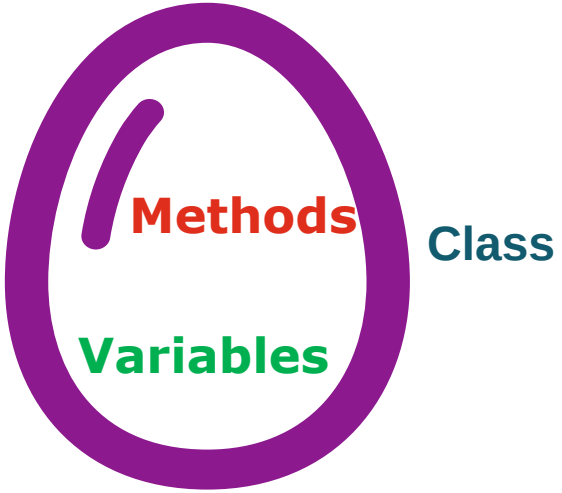
Designers

Note: An abstraction includes the essential details relative to the perspective of the user.

Encapsulation

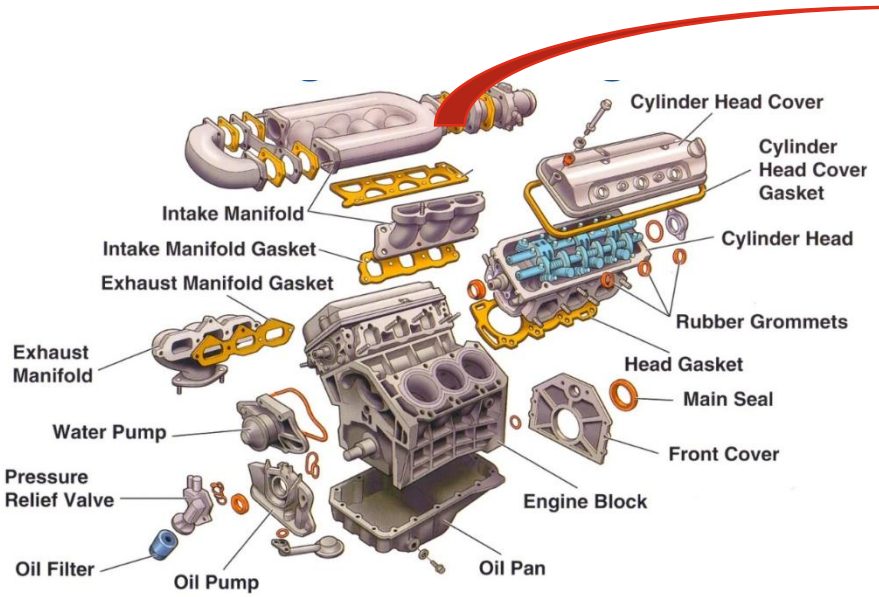
- Encapsulation refers to the process of **bundling the data (attributes or variables) and methods (functions or procedures)** that operate on the data into a single unit, called a **class**.
- It's like a capsule that protects the inner workings of an object. This helps keep code organized, makes it easier to understand, and prevents outside interference with the internal data.

Medicines are not directly accessible / protected from outside



Encapsulation

- All these methods and attributes are encapsulated within the car engine.
- This means that the internal workings of the engine are **hidden from the outside world**.



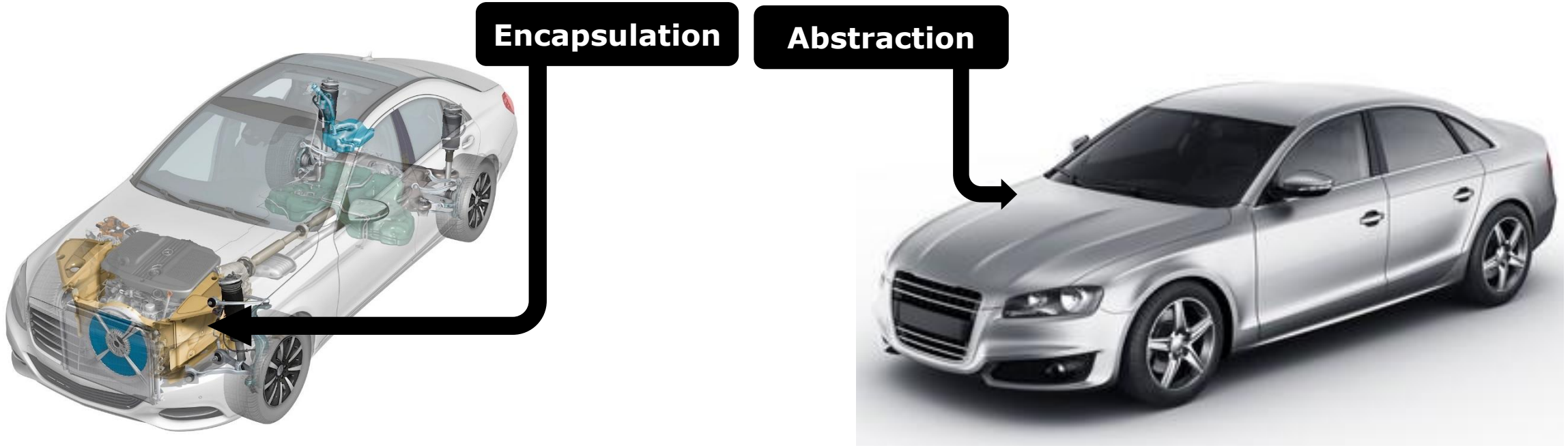
Note: **With encapsulation, the internal state of an object is hidden from the outside world, and interactions with the object are performed only through well-defined methods.**

Encapsulation - Real Life Example

- The car owner possesses the **key to the car that grants them full access**, allowing them to start the engine, unlock the doors, and access other features.
- Now, let's say someone else, like a neighbor or a friend, sees the car parked and wants to use it.
- However, since they don't have the key, they are **unable to access** the car or drive it.
- The car's lock and key system **restricts access to only the owner**.

Note: Encapsulation provides a level of access control, ensuring that only authorized users can interact with the class members.

Abstraction vs. Encapsulation



Note: Abstraction says **what details** to be made visible where as Encapsulation provides the **level of access** right to that visible details.

Inheritance

- Inheritance is a process in which **one object gets** the capability to acquire all the **properties and behaviours** of its parent object automatically.
- As a result, there is no need to rewrite the code again and again. This **'Code reusability'** feature avoids **code redundancy.**
- The **class** from which **properties are inherited** is called a **base class** and the **class** that **inherits** the **properties** is called a **derived class.**
- The derived class **uses** the **parent properties** and also **has its own properties**.

Inheritance

Grand Dad

House - owns

Inherited from

Dad

House - derives

Inherited from

Son

House - derives

Note:

Gen 1 — Granddad owns a house

Gen 2 — Dad **inherits** the house from grandad and **buys** a new car

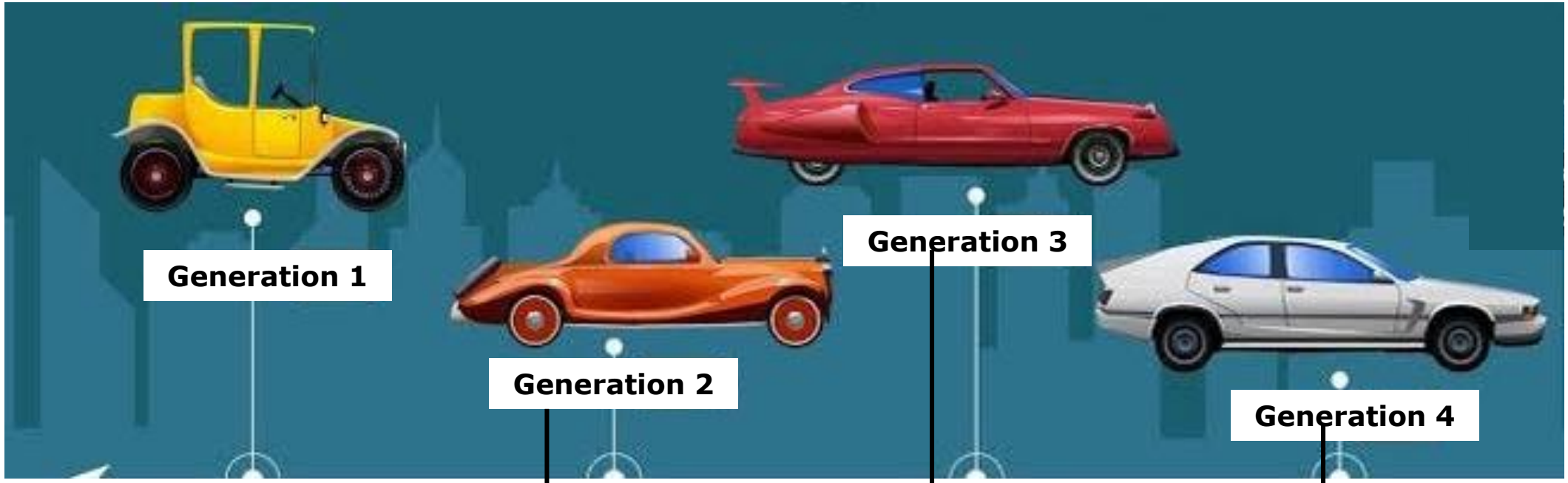
Gen 3 — Son **inherits** the house and car from previous generations and **buys** a new bike

Car - owns

Car - derives

Bike - owns

Inheritance – Real Life Example



Types of Inheritance

- There are 5 types of inheritance,

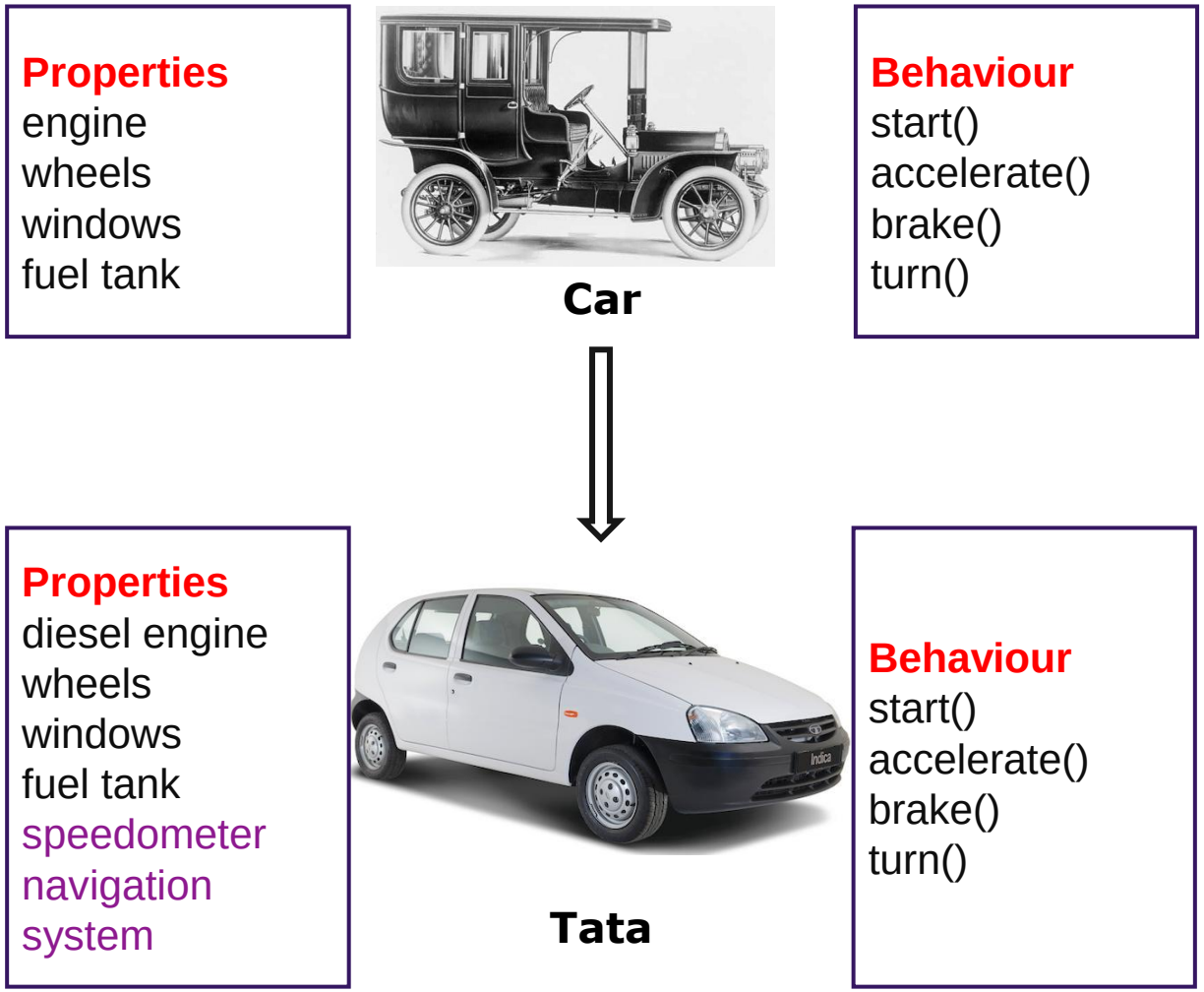
    1. Single level inheritance
    2. Multi-level inheritance
    3. Hierarchical inheritance
    4. Hybrid inheritance
    5. Multiple inheritance

Types of Inheritance

Single Level Inheritance

- In Single-level inheritance, there is a **single base class & a single derived class**
- i.e. - A base car feature is extended by the Tata brand and some new features are added.

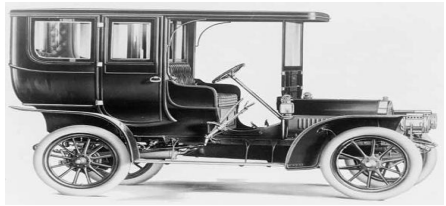
Properties from the base class is reused and new features / properties are added.



Types of Inheritance

Multi-Level Inheritance

- In Multilevel inheritance, there is **more than one single level of derivation.**
- i.e. - After base features are extended by the Tata brand. Now Tata brand has manufactured its new model with newly added features or advanced features like electric window, and touchscreen controls.
- From generalization, getting into more specification.



Car

Properties
engine
wheels
fuel tank

Behaviour
start()
accelerate()
brake()

Tata

Properties
speedometer

Behaviour

Tata Hexa

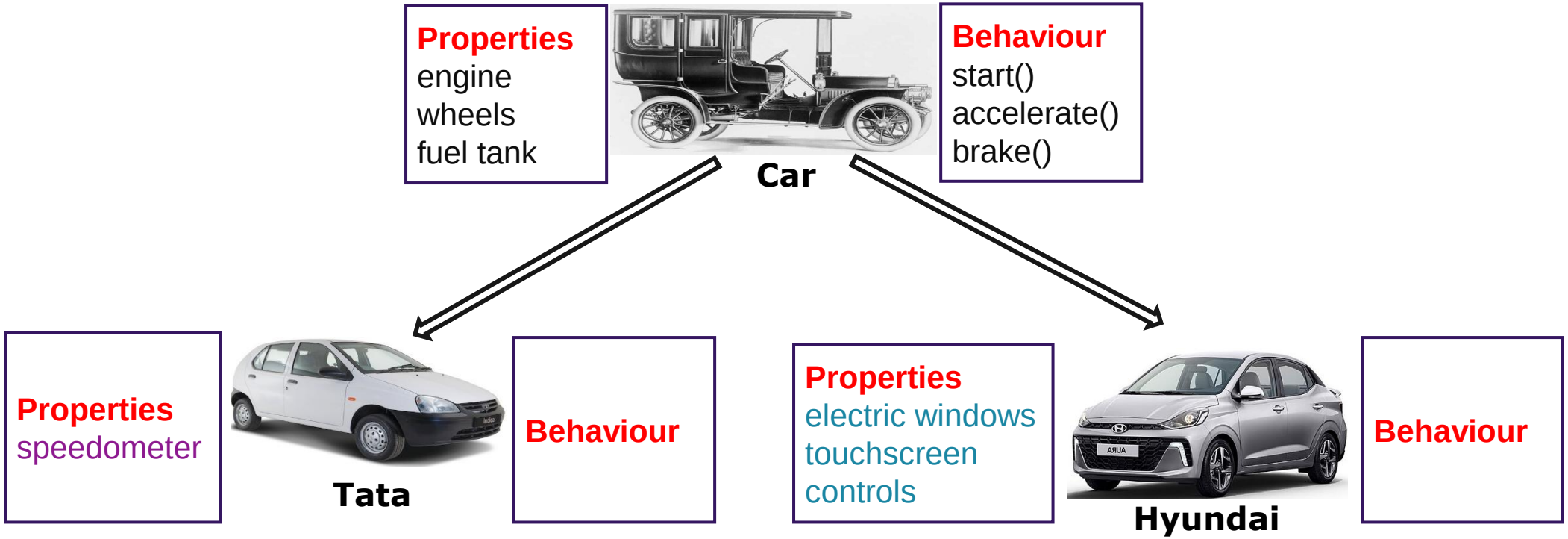
Properties
electric windows
touchscreen
controls

Behaviour

Types of Inheritance

Hierarchical Inheritance

- In this type of inheritance, **multiple derived classes would be extended from a base class**, it's similar to single-level inheritance but this time along with Tata, Hyundai is also taking part in inheritance.



Types of Inheritance

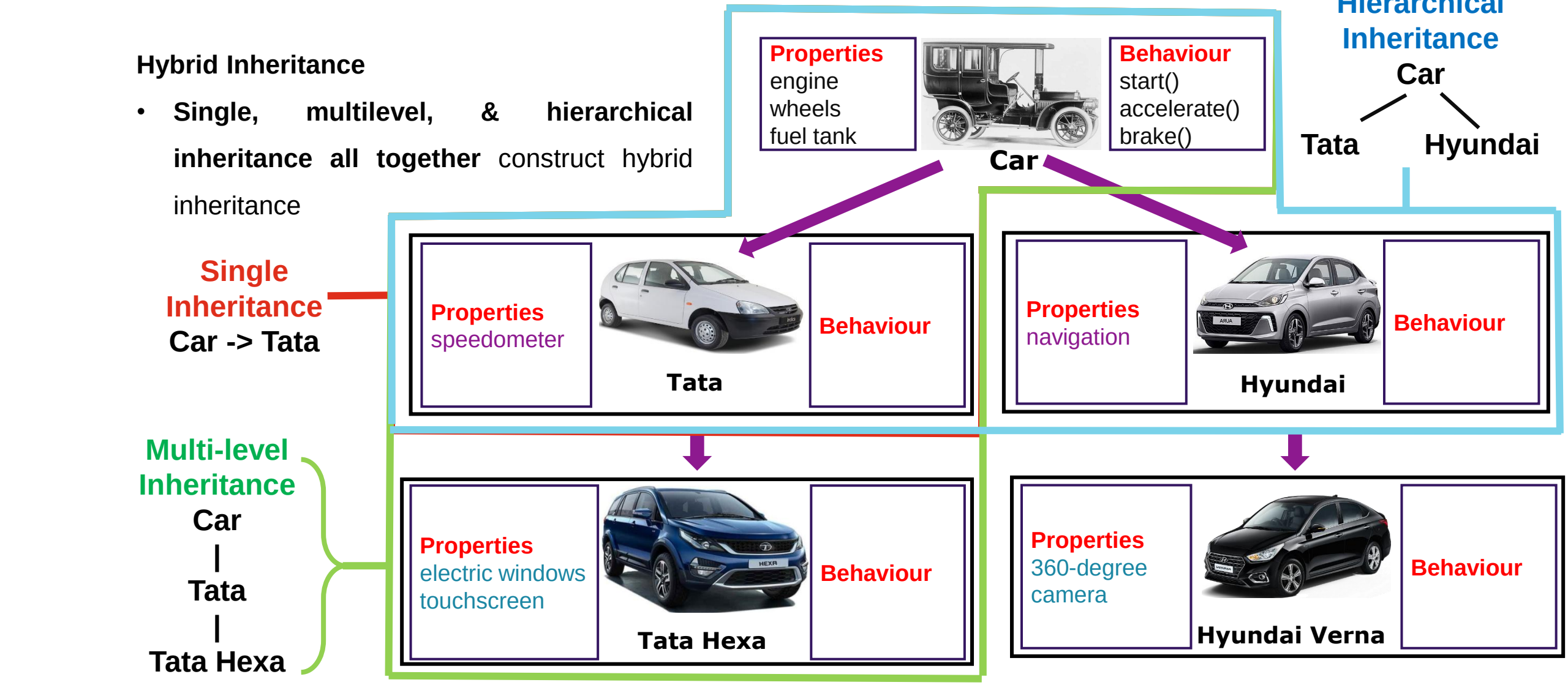
Hybrid Inheritance

- **Single, multilevel, & hierarchical inheritance all together** construct hybrid inheritance

Single Inheritance
Car -> Tata

Multi-level Inheritance
Car
|
Tata
|
Tata Hexa

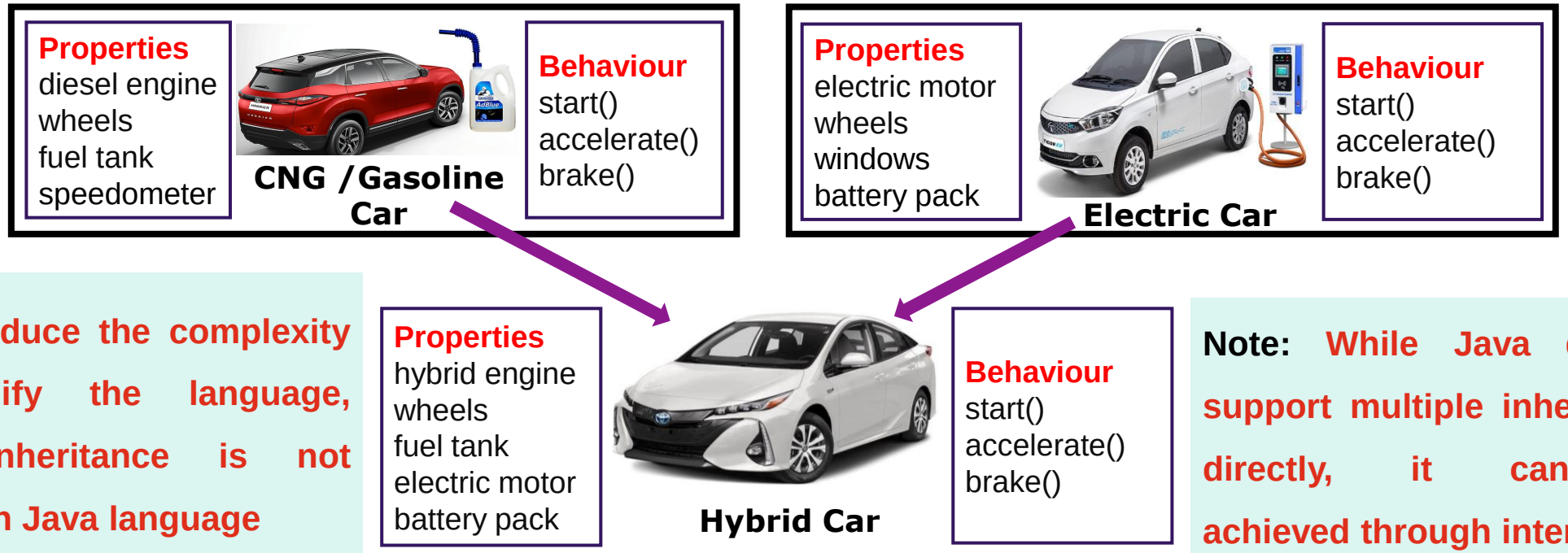
Hierarchical Inheritance
Car
Tata **Hyundai**



Types of Inheritance

Multiple Inheritance

- Multiple inheritance is where **derived class will extend from multiple base classes**.
- i.e. Tata will use the function of multiple Cars (Gasoline & Electric).



Note: **To reduce the complexity and simplify the language, multiple inheritance is not supported in Java language**

Note: **While Java doesn't support multiple inheritance directly, it can be achieved through interface**

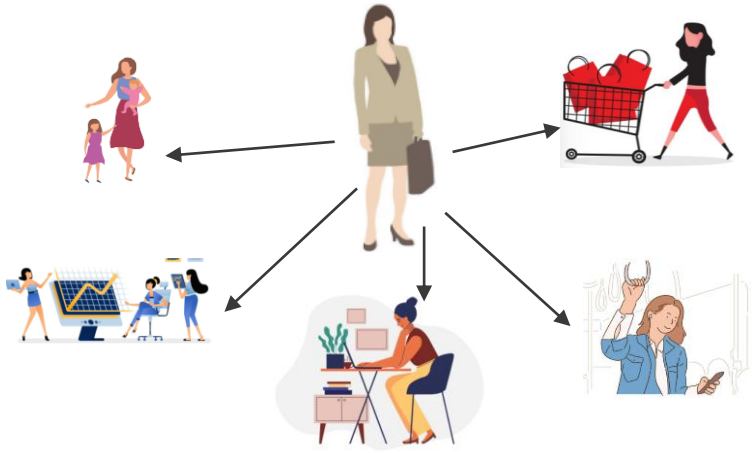
Polymorphism

- Polymorphism is the concept where an **object behaves differently in different situations.**
- More precisely we say it as **'many forms of single entity'.**

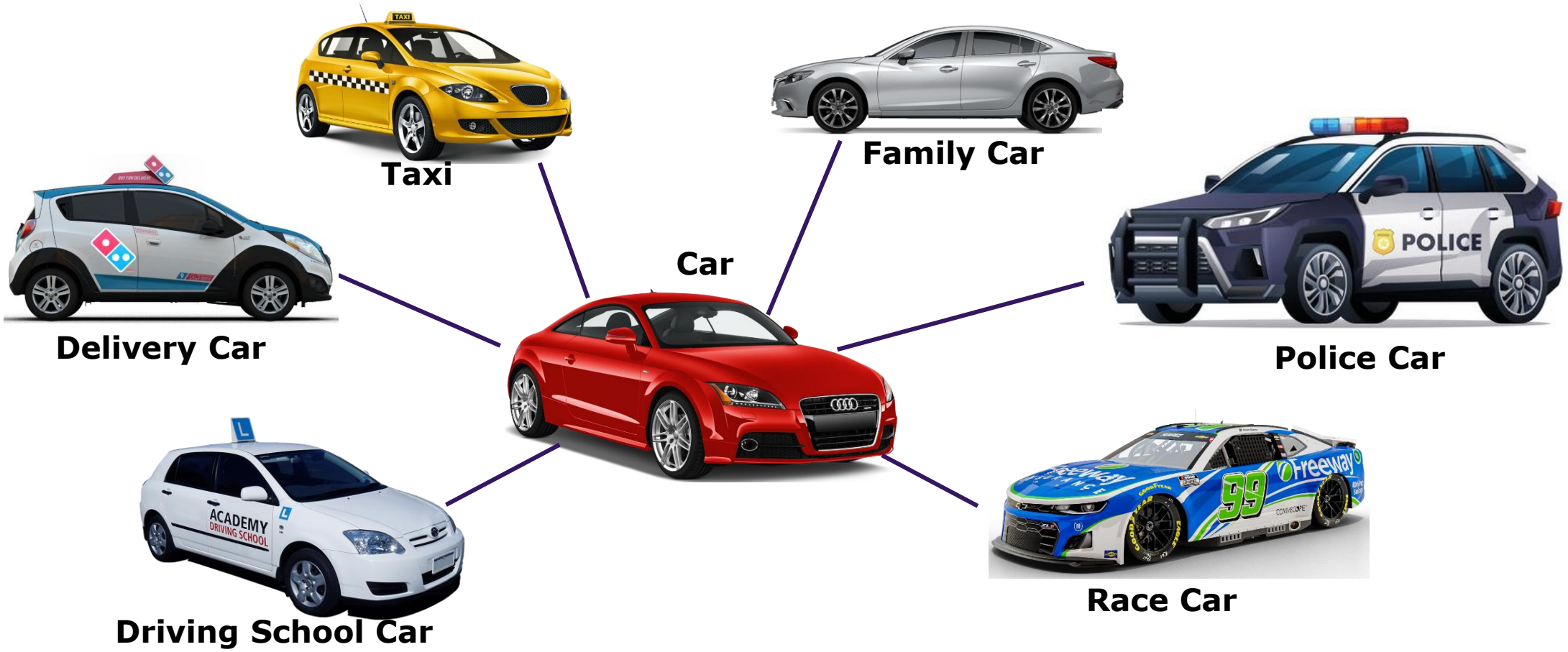
Polymorphism

Example 1: We turn on the computer by one button at the same time we can turn off the computer by the same button.

Example 2: A person might have a **variety of characteristics**. He is a man, and he can also be a father, a husband, or an employee. As a result, the same person behaves differently in different situations.

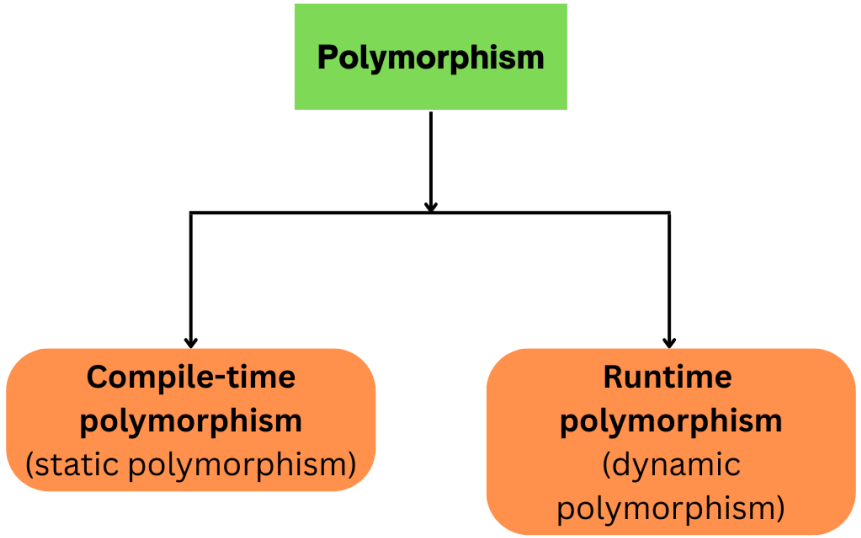


Polymorphism – Real Life Example



Note: One car behaves differently in different situations.

Types of Polymorphism



- **Compile-time polymorphism:** Decisions about which method to call are **made by the compiler**. Achieved through method overloading and operator overloading.
- **Runtime polymorphism:** Decisions about which method to call are **made at runtime, based on the actual type of the object**. Achieved through method overriding.

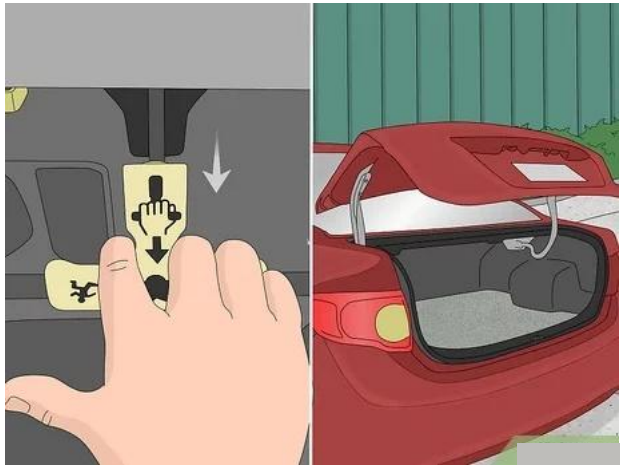
Types of Polymorphism

Method Overloading

- Method overloading is a feature in programming languages that allows you to define multiple methods/functions in the same class with the **same name but with different parameters**. This means you can have several methods in a class or module with the same name, but they perform different tasks based on the parameters they receive.
- It is one of the ways by which Java achieves Compile Time Polymorphism.

Types of Polymorphism

- Let's say a Car has a trunk i.e. – it is having the functionality of **OpenTrunk().** Now the same car has Electronic Trunk Release mode available in the car, so functionality would be the same but with different mode.
- The function OpenTrunk() remains the same, but there are two ways to achieve this **OpenTruck(key**) and **OpenTruck(ElectronicTrunkRelease)**



Types of Polymorphism

- This type is said to be Static polymorphism or Compile-time polymorphism.
- In our example -

Function	Parameter
OpenTrunk()	key
OpenTrunk()	ElectronicTrunkRelease

Note: **Same name different signature (parameters) – this is called Method Overloading**

Types of Polymorphism

Method Overriding

- Method overriding is a fundamental concept in object-oriented programming (OOP) where a subclass provides a specific implementation of a method that is already defined in its superclass, with the **same method signature**. This means that a subclass can redefine the behavior of a method inherited from its superclass to suit its own requirements.
- It is one of the ways by which Java achieves Run Time Polymorphism.

Types of Polymorphism

- **WiperOn()** was intended to turn on the wiper automatically when it rains, but suppose Tata had given provision to adjust the wiper manually in case of light rain - **Overriding the functionality to turn wiper on manually**.

Types of Polymorphism

- This type is called Dynamic polymorphism or Runtime polymorphism.
- In our example -

Function	Parameter
WiperOn()	RainIntensity
WiperOn()	RainIntensity

Note: **Same name same signature (parameters) – this is called Method Overriding**

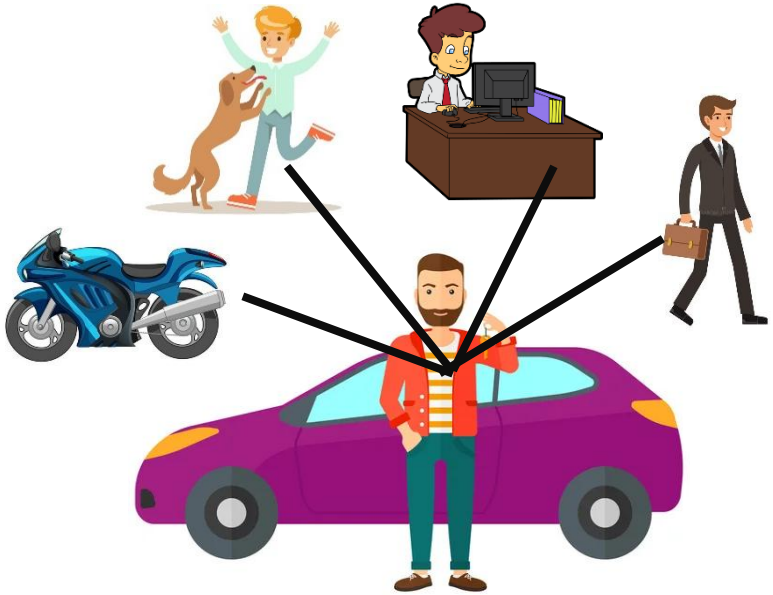
Association

- Association, aggregation, and composition are concepts related to **relationships between classes** in object-oriented programming. They are part of the broader concept of "object relationships" in OOP.
- **Association:** This is a more general relationship that **indicates how classes are related to each other**. It can be a one-to-one, one-to-many, or many-to-many relationship.

  For Example: A car has a driver. The relationship between the car and the driver is one of association. The car contains the driver,

  but the driver can also exist independently of the car. The driver can drive different cars or engage in other activities besides driving.

- There are **two types of Association**: Aggregation and Composition



Aggregation

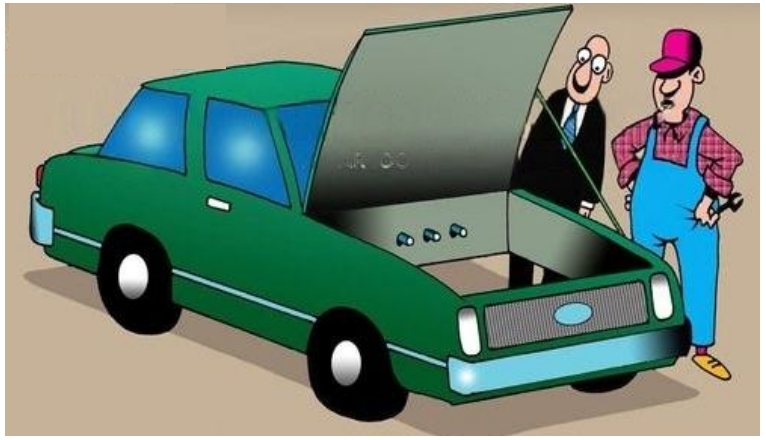
- **Aggregation:** This is a "has-a" relationship, where one class contains another class as a part, but the contained class can exist independently of the container class. It represents a **weaker form of ownership**. This is also called as **weak association**.

  For Example: A car has wheels. Each wheel is an independent object and can exist separately from the car. If you remove a wheel, it can still be used elsewhere, like in another car or as a spare.

Composition

- **Composition:** This is also a "has-a" relationship, but with **stronger ownership**. In composition, the contained class is part of the whole and cannot exist without the container class. This is also referred to as **strong association**.

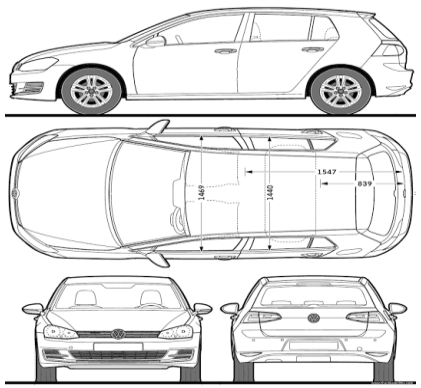
For Example: A car has an engine.



The engine is a crucial part of the car; without it, the car cannot function.

The engine is tightly coupled with the car, and if the car is destroyed, the engine is usually unusable in other contexts.

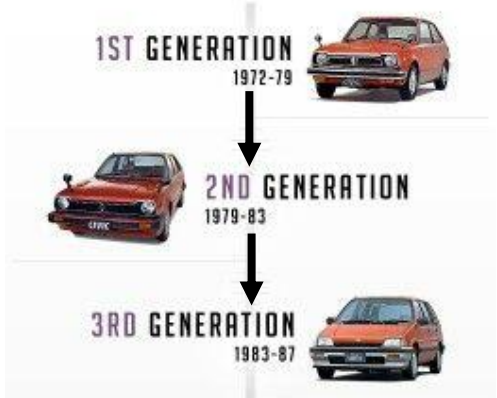
Summary: Car Object



CLASS

OBJECT

ENCAPSULATION



INHERITANCE

ABSTRACTION

POLYMORPHISM

Procedure Oriented Vs Object Oriented

POP	OOP
The program is divided into functions.	The program is divided into objects.
Top-down approach	Bottom-Up approach
Every function has different data, so there’s no control over it.	Data in each object is controlled on its own.
Parts of a program are linked through parameter passing.	Object functions are linked through message passing.
Expanding data and function is not easy.	Adding new data and functions is easy.
Inheritance is not supported.	Inheritance is supported in public, private& protected modes.

Procedure Oriented Vs Object Oriented

POP	OOP
No access modifiers supported.	Access control is done with access modifiers.
No data hiding. Data is accessible globally.	Data can be hidden using Encapsulation.
Overloading is not possible.	Overloading can be done.
No friend function.	No friend function except C++.
No virtual classes or functions.	The virtual function appears during inheritance.
No code reusability.	The existing code can be reused.
Not suitable for solving big problems.	Used for solving big problems.

Quiz

1) Identify equivalent OOP Principle for the following scenarios.

A person who understands more than two languages yet can converse in any of them depending on the situation.

Answer : Polymorphism

Quiz

2) Identify equivalent OOP Principle for the following scenarios.

A scientific calculator is a more advanced version of a regular calculator.

Answer : Inheritance

Quiz

3) Identify equivalent OOP Principle for the following scenarios.

When you go to the bank and tell the banker(Object) to make a deposit(Method) of X dollars into your account(object), the banker will ask for your account number as well as the amount of cash you want to deposit. After a few moments, the banker informs you. Hey, your deposit was completed successfully, and your receipt is attached.

Answer : Abstraction

Quiz

4) How many objects can you identify in the below picture?

a) 16

b) 2

c) 4

d) 8

Answer : a) 16

Quiz

5) **A Mobile phone comprises components such as a motherboard, camera, and sensors. The motherboard represents all the functions of a phone, the display shows the display only, and the phone is defined as a whole. Which of the following has the highest level of abstraction?**

a) Camera

b) Motherboard

c) Display

d) Mobile

Answer : d) Mobile

Quiz

6) Class is a collection of object

a) True

b) False

Answer : b) False

Quiz

7) Which of the following statement is correct?

a) Class is an instance of object.

b) Object is an instance of a class.

c) Class is an instance of data type.

d) Object is an instance of data type.

Answer : b) Object is an instance of a class.

Quiz

8) Which of the following concepts means wrapping up of data and functions together?

a) Abstraction

b) Encapsulation

c) Inheritance

d) Polymorphism

Answer : b) Encapsulation

”

Proper Preparation Prevents Poor Performance

- Charlie Batch


THANK YOU
SmartCliff
Career Mobility Solutions