OOP Introduction

What is OOP

- Object-Oriented Programming
- OOP is a programming paradigm
- It is based on the concept of objects
- An object contains attributes and behaviour
- Java is a pure object-oriented programming language

Class & Object

- An object is a real-world entity
- A class is a blueprint of an object, it is used to create an object
- An object has attributes and behaviours, In java they are defined as
  - Attribute -> variable
  - Behaviour -> methods

Car Example

- The car **has** mileage and colour (attributes)
- The car **does** drives, stops & give trip mileage (behaviour)
- Real world object (entity) of car can be alto10, bmw etc

```
public class Car {

    int mileage;

   String colour;

      public void drives() {

      }

      public void stops() {

      }

}


Car bmw = new Car();



      }
```

Animal Example

- An animal has colour, type, speed
- An animal does workType, rour

```
public class Animal {

 String colour;

    String type; // herbivour, carnivorous

    int speed;


    public void workType() {

    }


    public void rour() {


    }}. Animal cow = new Animal();
```

Constructor

- A **constructor** is a special method in a class.
- It is used to **initialise objects.**
- It is automatically called **when an object is created.**
- Key points
    A. Has the same name as the class.
    B. Has no return type

This

- This is a reference variable in Java that refers to the **current object**
- Uses
  - To access the current object’s instance variables.
  - To call current class methods.
  - To call another constructor.
  - To pass the current object as an argument.
  - To return the current object.

Static

- Static is a keyword in Java.
- Shared with all objects
- Uses
  - Static variable -
    - You want some data/method that is **common** for all objects.
  - Static method -
    - You want to call a method **without creating an object**.
  - Static block -
    - You want to initialise something **exactly once**.
  - Static class (only inner class) - no need for outer object.

- https://play.kahoot.it/v2/*?quizId=63c8e941-af3d-4058-ba7e-12958d926818

Pillars of OOP

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction

1. Encapsulation

Encapsulation is the mechanism of **wrapping data (variables) and code (methods) into a single unit called a class** and hiding data using **access modifiers**.

Syntax

- Create class
  - <access modifier> class <name of class> { wrap data/variable and methods/function }
- Creating variable
  - <access modifier> <data_type> <variable name> = <value>;
- Creating method
  - <access modifier> <return type> <function name> { … }

Type of Access Modifier

- public
- protected
- default <we do not mention default>
- private

Visibility of Access Modifier

Modifier	Same Class	different class	different package	different project
public	✓	✓	✓	✓
protected	✓	✓	X	✗
default	✓	X	✗	✗
private	✓	✗	✗	✗

2. Inheritance

Inheritance is a process where one class acquires the properties (methods and attributes) of another class.

IS-A vs HAS-A Relationship

IS-A vs HAS-A Relationship

1️⃣ IS-A (Inheritance)

- Represents **parent-child relationship**
- Uses **extends**
- **Car IS-A Vehicle**, so Car inherits properties of Vehicle

java

```java
class Vehicle { }
class Car extends Vehicle { }
```

✔ Used for **code reuse** and **polymorphism**

2️⃣ HAS-A (Composition / Aggregation)

- Represents **ownership or association**
- One object **contains** another object
- **Car HAS-A Engine**

java

```java
class Engine { }
class Car {
    private Engine engine;
}
```

✔ Used for **modular design** and **strong abstraction**

Example

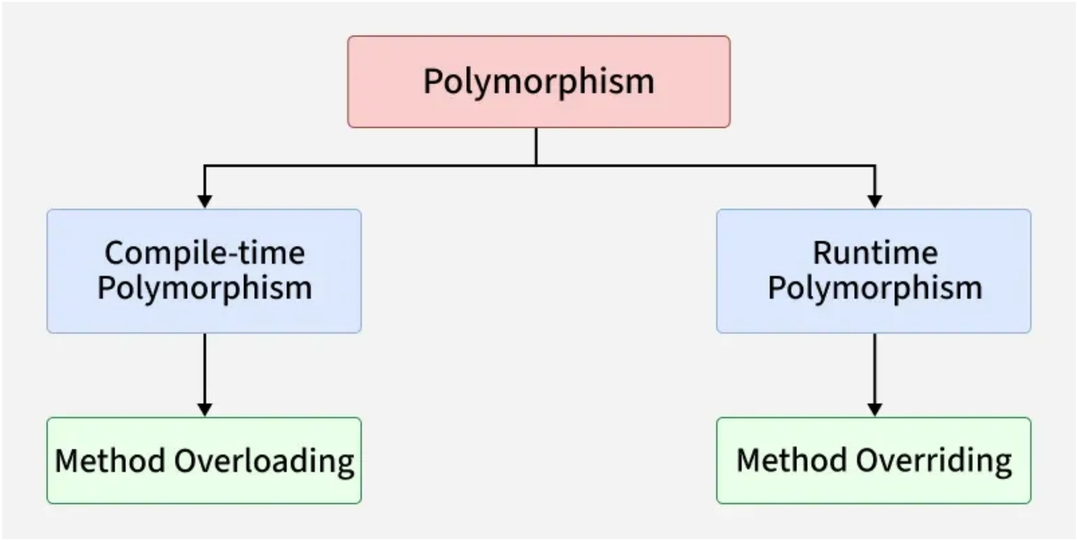
Real life examples

Real world	OOP Meaning
A BMW is a Car	IS-A
A Laptop has a Keyboard	HAS-A
A Student is a Person	IS-A
A House has a Door	HAS-A

3. Polymorphism

Polymorphism is the ability of an object to take on many forms

Type of polymorphism



Compile Time Polymorphism

It is also Static Polymorphism

This type is achieved through **method overloading**.

1. **Method Overloading** occurs when multiple methods in the **same class** have the **same name** but different **parameter lists**

2. The compiler decides which method to call at **compile time** based on the method signature (name and parameters).

Run-time Polymorphism

It is also called Dynamic Polymorphism

This type is achieved through **method overriding and inheritance**.

1. **Method Overriding** occurs when a **subclass (child class)** provides a specific implementation for a method that is already defined in its **superclass (parent class)**.

2. The method signature (name and parameters) must be **identical** in both the parent and child classes.

3. The decision of which method to call is made at **run time** based on the actual object type, not the reference type.

Abstraction

Abstraction is a process of **hiding** the implementation details from the user

Key Concepts of Abstraction

- Abstraction focuses on what an object does, rather than how it achieves it
- Two ways to achieve abstraction
    1. Abstract class
    2. Interface

Abstract Class

- **Definition:** A class declared with the **abstract** keyword. It cannot be instantiated (you cannot create an object of an abstract class).
- **Purpose:** To be extended (inherited) by other classes.
  - It can have **abstract methods** (methods with no body, just a signature, ending with a semicolon), and must be implemented by a subclass.
  - It can also have concrete (non-abstract) methods with full implementation

Interface

- **Definition:** A blueprint of a class. All methods declared in an interface are implicitly public and abstract.
- **Purpose:** To define a contract for classes that implement it.
  - With interface we achieve 100% abstractions
  - we used implements keyword to implement interface