PROGRAMMING PARADIGM

A programming paradigm is a fundamental style or approach of a programming that defines the way a programmer structures and develops code. Each paradigm provides a distinct perspective on how to solve problems and structure solutions.

It provides a set of principles, concepts, techniques for designing and implementing programs, structure and flow of the code as well as methodologies.

Here are some of the most common programming paradigms:

1. Procedural Programming

- **Description**: Focuses on a sequence of instructions or procedures to be followed. The program is divided into functions, and data is passed between these functions.
- **Languages**: C, Pascal, Fortran
- **Key Concepts**: Functions, procedures, modularization, sequence, iteration, and recursion.

2. Object-Oriented Programming (OOP)

- **Description**: Organizes code into objects that contain both data and methods that operate on the data. It emphasizes the concepts of classes, inheritance, polymorphism, and encapsulation.
- **Languages**: Java, C++, Python, Ruby
- **Key Concepts**: Classes, objects, inheritance, encapsulation, polymorphism, abstraction.

Classes

- **Definition**: A class is a blueprint or template for creating objects. It defines a set of attributes (data) and methods (functions) that the objects created from the class will have. Classes encapsulate data for the object and define behaviors that can operate on that data.

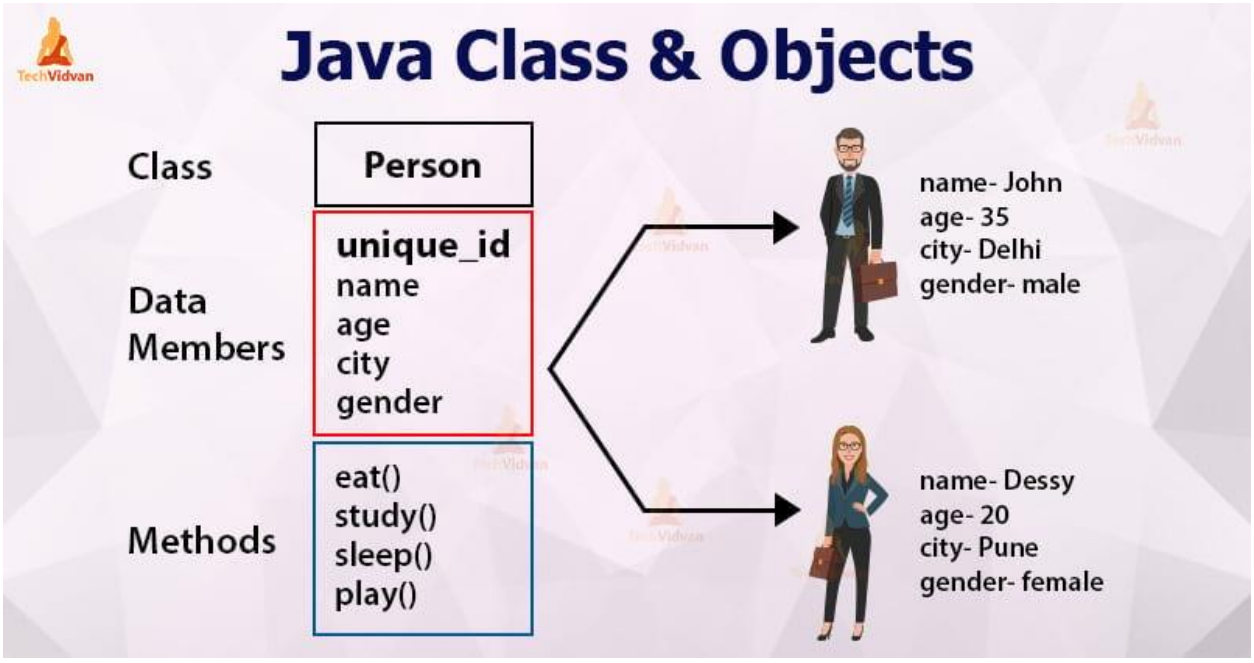
Objects

- **Definition**: An object is an instance of a class. When a class is defined, no memory is allocated until an object of that class is created. Objects can have unique values for their attributes, even if they are instances of the same class. The object represents the real-world entity, and the class is the abstraction.

Class	Object

Compiled by : Er.Gaurab Mishra
Computer department (HOD)
K.M.C COLLEGE

Class is the blueprint of an object. It is used to declare and create objects.	Object is an instance of class.
No memory is allocated when a class is declared.	Memory is allocated as soon as an object is created.
A class is a group of similar objects.	Object is a real-world entity such as book, car, etc.
Class is a logical entity.	Object is a physical entity.
Example of class can be car.	Objects of the class car can be BMW, Mercedes, jaguar, etc.



Compile Time vs. Run Time

Compiled by : Er.Gaurab Mishra
Computer department (HOD)
K.M.C COLLEGE

1. **Definition:**
    - **Compile Time:** The period during which source code is translated into executable code by a compiler. It involves checking for syntax errors, type checking, and converting high-level code into machine code.
    - **Run Time:** The period during which the compiled code is executed by the computer's processor. It involves the actual execution of the program instructions.

Types of polymorphism

Compile-Time Polymorphism, also known as **Static Polymorphism**, is a type of polymorphism in which the method to be invoked is determined during the compilation of the program. This is achieved through two main techniques: **method overloading** and **operator overloading**.

Key Techniques of Compile-Time Polymorphism

1. Method Overloading

- **Definition**: Method overloading allows a class to have more than one method with the same name, as long as their parameter lists (number of parameters or types of parameters) are different. The compiler determines which method to call based on the method signature (name + parameter types) at compile time

2. **Operator Overloading**
Operator overloading allows developers to define custom behavior for operators (like `+`, `-`, `*`, etc.) when they are used with user-defined types (e.g., classes). This is primarily supported in languages like C++.

Runtime Polymorphism, also known as **Dynamic Polymorphism**, is a type of polymorphism where the method to be invoked is determined at runtime, rather than at compile time. This is achieved primarily through **method overriding**, and it allows for more flexible and dynamic code, especially in the context of inheritance and object-oriented design.

Access specifiers

Access specifiers in Object-Oriented Programming (OOP) are keywords that define the accessibility or scope of a class's members (i.e., attributes and methods). They control how the members of a class can be accessed from other parts of the code. The three primary types of access specifiers are:

1. Public

- **Definition**: Members declared as `public` can be accessed from any part of the program. There are no restrictions on accessing public members.

Compiled by : Er.Gaurab Mishra
Computer department (HOD)
K.M.C COLLEGE

2. Private

- **Definition**: Members declared as `private` can only be accessed within the class in which they are declared. They are not accessible from outside the class, not even by derived classes.

3. Protected

- **Definition**: Members declared as `protected` can be accessed within the same class, by derived classes (subclasses), and by classes within the same package (in languages like Java). However, they are not accessible from outside these contexts

- **Public**: Members can be accessed from anywhere.
- **Private**: Members can only be accessed within the class itself.
- **Protected**: Members can be accessed within the class and by derived classes.

Compiled by : Er.Gaurab Mishra
Computer department (HOD)
K.M.C COLLEGE