UNIT I

Programming paradigms

Programming paradigms are different styles or approaches to writing computer programs. Each paradigm provides a way of thinking about problems and structuring solutions.

Paradigm	Focus	Examples
Procedural	Functions & steps	C, Pascal
OOP	Objects & classes	Java, C++
Functional	Pure functions	Haskell, Scala
Logical	Rules & facts	Prolog
Event-Driven	Events & handlers	JavaScript
Concurrent	Parallel tasks	Go, Java
Scripting	Quick automation	Python, Bash
Declarative	What to do, not how	SQL, HTML

Difference Between Structured Programming and OOP

Feature	Structured Programming	Object-Oriented Programming (OOP)
Basic Approach	Follows a **top-down** approach	Follows a **bottom-up** approach
Program Structure	Program is divided into **functions**	Program is divided into **objects and classes**
Focus	Emphasizes on **functions and procedures**	Emphasizes on **objects and data**
Data Handling	Data is **not well protected**; any function can access data	Data is **encapsulated** inside objects, improving security

Feature	Structured Programming	Object-Oriented Programming (OOP)
Reusability	Limited code reuse	High code reuse through **inheritance** and **polymorphism**
Example Languages	C, Pascal	Java, C++, Python
Real-World Representation	Difficult to represent real-world entities	Easy to model real-world objects (Car, Employee, etc.)
Data & Function Relationship	Data and functions are **separate**	Data and functions are **combined** inside classes
Maintenance	Harder to maintain large programs	Easier to maintain and extend due to modularity
Flexibility	Less flexible for complex systems	Highly flexible for large and complex projects
Security	Low security	High security with access modifiers (private, public, protected)

Structured Programming

- Breaks programs into functions.
- Focuses on step-by-step instructions.
- Good for simpler tasks.
- Example: C language.

Object-Oriented Programming

- Breaks programs into classes and objects.
- Focuses on representing real-world objects.
- Good for large and complex applications.
- Example: Java, Python, C++.

NEED FOR OBJECT ORIENTATION

Manage Increasing Software Complexity

- Modern applications are large and complex.
- OOP breaks them into **objects** (small, manageable units).

To Model Real-World Problems Easily

- Real-world entities (Car, Employee, Bank Account) can be directly represented as **objects**.
- This makes software design more natural and intuitive.

To Improve Code Reusability

- Using **inheritance**, you can reuse existing code instead of rewriting it.
- Saves time and effort during development.

To Enhance Maintainability

- Each object is independent; modifying one does not affect others.
- Makes debugging and updating easier.

To Provide Better Data Security (Encapsulation)

- Data is hidden inside objects using **private** and **protected** access.
- Prevents accidental or unauthorized access to important data.

To Support Flexibility With Polymorphism

- Same function can behave differently for different objects.
- Makes programs more flexible and extensible.

To Promote Modularity

- Programs are divided into various **classes and objects**.
- Easy to divide work among team members.

To Reduce Development Time

- OOP provides reusable libraries, frameworks, and components.
- Faster and more reliable development.

Useful for Large-Scale & Long-Term Projects

- Banking systems, ERPs, real-time simulations, gaming engines, etc.
- OOP helps in building and maintaining huge systems over many years.

Object-Oriented Programming (OOP) **Principles**

Object-Oriented Programming (OOP) is built on **four fundamental principles**. These principles help create scalable, secure, and maintainable software.

Abstraction

Abstraction in Object-Oriented Programming (OOP) is the process of **showing only the essential details** of an object while **hiding the unnecessary internal complexity**.

What is Abstraction?

Abstraction focuses on **what an object does** rather than **how it does it**.

Shows only important features

- **Hides internal implementation**
- **Reduces complexity**
- **Improves security and usability**

Example (Real Life)

Think of a **car**:

- You see the steering wheel, pedals, gear.
- You *do not see* the engine mechanism, fuel injection, wiring, etc.

You use the car **without knowing the internal details** → That's abstraction!

Abstraction in OOP – Key Points

- **Hides complexity from the user**
- **Helps focus on what the object can do**

Achieved using:

- **Abstract classes**
- **Interfaces**

Encourages cleaner, simpler code

Encapsulation

Encapsulation is one of the core principles of Object-Oriented Programming (OOP).
It refers to **bundling data and methods** that operate on that data **within a single unit (class)** and **restricting access** to some components.

What is Encapsulation?

Encapsulation means:

- **Wrapping data (variables) and code (methods) into one unit**
- **Controlling access to data using access modifiers**
- **Protecting data from unauthorized access or modification**

It is also known as **data hiding**.

Why Encapsulation Is Important

- Protects the data from accidental changes
- Gives control over how data is accessed or modified
- Increases security
- Makes code modular and maintainable

How Encapsulation Works

Encapsulation is achieved using:

- **Private variables (data is hidden)**
- **Public getter and setter methods (controlled access)**

Encapsulation	Abstraction
Hides data	Hides complexity
Protects data	Shows only essential features
Achieved using: private variables + getters/setters	Achieved using abstract classes/interfaces

Inheritance

Inheritance is one of the fundamental principles of Object-Oriented Programming (OOP).
It allows one class (**child/subclass**) to **acquire the properties and behaviors** of another class (**parent/superclass**).

What Is Inheritance?

Inheritance means:

- **Creating a new class from an existing class**
- **Reusing code instead of rewriting it**
- **Establishing a parent–child relationship between classes**

The new class **inherits**:

- Variables (attributes)
- Methods (functions)

Why Inheritance Is Needed

- Promotes **code reusability**
- Improves **maintainability**
- Supports **polymorphism**
- Helps build **hierarchical (family-like) structures**

Simple Example (Real Life)

A **Dog** is an **Animal**.
So, Dog inherits common features of Animal (e.g., breathe, eat) and adds its own (bark).

Polymorphism

Polymorphism is one of the most powerful principles of Object-Oriented Programming (OOP).
It means **"one name, many forms"** — the same method or function behaves differently depending on the object or situation.

What Is Polymorphism?

Polymorphism allows:

- **One function name → multiple behaviors**
- **Same method → different implementations**
- **Flexibility and extensibility in code**

Types of Polymorphism

Compile-Time Polymorphism (Static)

Achieved using **method overloading**.

Method Overloading:

Multiple methods with the **same name** but **different parameters**.

Run-Time Polymorphism (Dynamic)

Achieved through **method overriding**.

Method Overriding:

A child class provides a **new implementation** for a method already defined in the parent class.

Why Polymorphism Is Useful

- Makes code **flexible**
- Supports **extensibility** (easy to modify or add features)
- Enables **method overriding** in inheritance
- Improves **readability and reusability**

Real-Life Example

A **remote control** (method name) can operate:

- TV
- AC
- Sound system

Same button → different results depending on the device.
This is **polymorphism**.

Simple Summary

Type	How It Works	Example
Compile-Time Polymorphism	Method overloading	Same method name, different parameters
Run-Time Polymorphism	Method overriding	Child class changes parent method

Easy Definition to Remember

Polymorphism = One interface, multiple implementations

Classes and Objects in OOP

CLASS in OOP

A **class** is a **blueprint/template** used to create objects.

It defines:

- **Data (variables/attributes)**
- **Functions (methods/behaviors)**

A class does *not* occupy memory until an object is created.

OBJECT in OOP

An **object** is an **instance of a class**.

It represents a real-world entity

It has:

- **State** (values of attributes)
- **Behavior** (methods it can perform)

Each object has its own copy of the class properties.

Relationship Between Class and Object

Class	Object
Blueprint	Actual item
Definition	Instance
No memory until object created	Uses memory
Describes properties	Has actual values

Real-Life Example

Class: Mobile Phone

Attributes → brand, price, color
Methods → call(), message(), camera()

Object:

- Samsung phone → brand="Samsung", price=15000
- iPhone → brand="Apple", price=70000

Both are objects of the **Mobile Phone** class.

Why Classes and Objects Are Important

- Help model real-world things
- Provide modularity
- Increase reusability
- Form the foundation of OOP

Simple Definitions to Remember

Class = Blueprint

Object = Instance of the class

Message Passing

Message Passing is an important concept in Object-Oriented Programming (OOP).
It describes **how objects communicate with each other**.

What is Message Passing in OOP?

Message passing means:

- **One object sends a message (request) to another object**
- **The receiving object responds by executing a method**
- **It is the way objects interact in OOP**

In simple words:
Object A asks Object B to perform an action by calling its method.

How Message Passing Works

1. **Object (Sender)** → sends a message (method call)
2. **Object (Receiver)** → receives the message
3. Receiver object runs the requested method and returns a result

Example:

Benefits of Message Passing

- Promotes **interaction** between objects
- Supports **encapsulation** (communication only through methods)
- Makes programs modular and maintainable
- Helps achieve **loose coupling** (objects depend less on each other)

Real-Life Example

When you press a button on a **TV remote**:

- Remote sends a **message** to the TV
- TV receives the message and performs an action (increase volume, change channel)

Same in OOP: Objects communicate by **sending messages** (method calls).

Simple Definition to Remember

Message Passing = Sending methods as messages between objects to communicate and perform actions.

Advantages and **Applications** of Object-Oriented Programming (OOP)

ADVANTAGES OF OOP

1. Modularity

Programs are divided into classes and objects, making the code easy to manage, understand, and update.

2. Reusability

Features like **inheritance** allow you to reuse existing code, reducing duplication and saving time.

3. Data Security

Encapsulation protects data by restricting direct access through private variables and public methods.

4. Flexibility

Polymorphism allows the same function name to behave differently, making code easy to extend and modify.

5. Easy Maintenance

OOP code is easier to maintain because:

- Classes are independent
- Changes in one class don't affect others much
  This reduces debugging time.

6. Real-World Modeling

OOP models real-world entities as objects, making program design natural and intuitive.

7. Improved Productivity

Reusability, modularity, and clarity lead to faster development.

8. Scalability

OOP systems can be expanded easily by adding new classes and objects without changing existing code.

APPLICATIONS OF OOP

1. Software Development

Used in designing:

- Operating systems
- Compilers
- Text editors

- Business applications

2. Graphical User Interfaces (GUIs)

GUI elements like buttons, windows, icons are modeled as objects.
Used in:

- Java Swing
- .NET Framework
- Android development

3. Game Development

Games use many objects (players, enemies, obstacles).
OOP helps handle their behavior and interaction.

4. Web Applications

Languages like Java, Python, PHP, and JavaScript (with OOP support) are used to create dynamic web apps.

5. Real-Time Systems

Used in:

- Traffic control systems
- Sensor monitoring
- Robotics
  Objects help interact with hardware efficiently.

6. Simulation & Modeling

Used for:

- Flight simulators
- Weather models
- Scientific calculations

OOP supports complex system modeling.

7. Database Systems

Object-oriented database management systems (OODBMS) store data as objects.

8. Mobile Application Development

Android (Java/Kotlin) and iOS (Swift) use OOP extensively.

In Simple Words…

Advantages:

OOP makes programs **modular, reusable, secure, scalable, and easy to maintain**.

Applications:

Used in **software, games, apps, GUIs, simulations, databases, and real-time systems**.