**Inheritance** when a new class can inherit **attributes** and **behaviors** from an existing class. This allows for code reuse and the creation of **hierarchical relationships** between classes. Inherited properties can be extended or overridden in the **subclass**.

**Polymorphism** enables objects of different classes to be treated as objects of a common superclass. This promotes **flexibility** and **extensibility** in code. Polymorphism is achieved through method **overriding** and method **overloading**.

**Encapsulation** involves **bundling data** (attributes) and **methods** (functions) that operate on the data into a **single unit**, called a class. Access to the data is controlled through methods, **ensuring data integrity** and **hiding implementation details**.

**Abstraction** focuses on presenting **essential features** of an object while **hiding unnecessary details**. It allows for the creation of **abstract classes** and **interfaces** that define a **blueprint** for subclasses. Abstraction helps in designing **modular** and **loosely coupled** systems.

An **interface** in Java is like a **blueprint for a class**. It defines a set of methods that a class **must implement**. Think of it as a contract that a class agrees to follow. When a class **implements** an interface, it promises to provide specific functionality that the interface outlines. This allows different classes to share **common methods** without being tightly connected in terms of inheritance.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Abstract class vs Interface**

* An abstract class is a class that provides some functionality and enforces certain rules but **leaves some parts for its subclasses to complete**;
* An interface, on the other hand, is like a checklist that a class needs to fulfill, specifying **what methods it must have**, without providing any actual code.