**1. Polymorphism in Object-Oriented Programming**

**Concept of Polymorphism:**

Polymorphism is the ability of different classes to be treated as instances of the same class through inheritance. Specifically, polymorphism allows a method or function to behave differently based on the object that it is operating on. This can occur in two main forms:

* **Method Overriding** (runtime polymorphism): A subclass can provide a specific implementation of a method that is already defined in its superclass.
* **Method Overloading** (compile-time polymorphism): A class can have multiple methods with the same name but different parameter lists.

**How It's Represented in a Class Diagram:**

Polymorphism is represented through **inheritance** and **method overriding** in a class diagram. Here's how it's typically shown:

* A base class has a method signature.
* Derived classes inherit from the base class and override the base class method, providing their own implementation.

Example:

* Suppose we have a base class LibraryItem with a method getDetails(). Then, Book and Magazine can inherit from LibraryItem and override getDetails() to return book details or magazine details, respectively.

+-------------------+ +-------------------+ +-------------------+

| LibraryItem |<----| Book |<----| Magazine |

+-------------------+ +-------------------+ +-------------------+

| + getDetails() | | + getDetails() | | + getDetails() |

+-------------------+ +-------------------+ +-------------------+

* **LibraryItem** is the base class, and both **Book** and **Magazine** are derived classes that override the getDetails() method.
* This enables polymorphism: when the getDetails
* () method is called on a LibraryItem reference, it will invoke the correct method depending on whether it's a Book or Magazine.

**2. Difference Between an Association and a Dependency Relationship**

In a **class diagram**, both **association** and **dependency** are types of relationships, but they differ in their nature and strength:

**Association:**

* **Definition**: An association represents a **structural relationship** between two classes, where objects of one class are connected to objects of another. It's a **stronger** relationship compared to a dependency.
* **Characteristics**:
  + It indicates that one class is aware of the other, and they usually interact with each other.
  + Can be **bi-directional** or **uni-directional**.

**Example**: In a library system, a **Book** might have an association with a **Loan**. A book can be loaned out, and this relationship is structural.

**Dependency:**

* **Definition**: A dependency represents a **weaker relationship** where one class **depends** on another class for some behavior or operation, but it does not directly control or own the other class. It's often a **temporary** relationship.
* **Characteristics**:
  + It shows that one class needs another to function, but does not maintain a long-lasting relationship.
  + Often used for **method calls** or **parameter passing**.
  + Represented by a dashed arrow, where the arrow points from the dependent class to the class it depends on.

**Example**: In a library system, a **Member** might depend on a **Book** to borrow it, but it doesn’t own or directly interact with the Book object in a structural way