Let's explore these Java concepts with examples from a Library Management System (LMS), using easy-to-understand analogies and code snippets.

1. Inheritance

Inheritance allows one class (child) to inherit the properties and behaviors of another class (parent). Imagine a "Library" is a parent class, and specific sections like "Fiction" or "Non-Fiction" are the child classes. The child inherits some common behaviors but can have its own specific methods too.

Example:

- Parent Class: LibraryItem
- Child Class: Book, Magazine

Code: java

```
// Parent class
class LibraryItem {
    protected String title;
    protected String author;
// Constructor
    public LibraryItem(String title, String author) {
        this.title = title;
        this.author = author;
    }
// Method to display details
    public void displayInfo() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
    }
}
// Child class (Book)
class Book extends LibraryItem {
    private int pageCount;
// Constructor
    public Book(String title, String author, int pageCount) {
```

```
        super(title, author);  // Calling the parent constructor
        this.pageCount = pageCount;
    }

// Method to display book-specific details
    @Override
    public void displayInfo() {
        super.displayInfo();  // Calling the parent method
        System.out.println("Pages: " + pageCount);
    }
}
// Child class (Magazine)
class Magazine extends LibraryItem {
    private int issueNumber;

    public Magazine(String title, String author, int issueNumber) {
        super(title, author);
        this.issueNumber = issueNumber;
    }
    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Issue Number: " + issueNumber);
    }
}
```

Layman Analogy:

Imagine a parent as a general "blueprint" (like a car). The child (a specific car model) inherits the basic structure (wheels, engine) but can add specific features (like color, speed).

Access Modifiers:

- public: Can be accessed by any class.

- protected: Accessible within the package and subclasses.
- private: Only accessible within the class.

Code: java

```
class LibraryItem {
   public String title;     // Accessible anywhere
   protected String author; // Accessible to subclasses
   private int id;          // Only accessible within LibraryItem class
}
```

Method Overloading:

Method overloading allows multiple methods with the same name but different parameters. It's like asking a librarian for help with either a book title or an author.

Example:

java

```
class LibraryItem {
   public void search(String title) {
      System.out.println("Searching by title: " + title);
   }
   public void search(String title, String author) {
      System.out.println("Searching by title and author: " + title + ", " + author);
   }
}
```

Multilevel Inheritance:

In multilevel inheritance, a class is derived from a class that is also derived from another class.

Example:

java

```
// Level 1: Parent class
class LibraryItem {
   protected String title;
   protected String author;
```

```
    public LibraryItem(String title, String author) {
        this.title = title;
        this.author = author;
    }
    public void displayInfo() {
        System.out.println("Title: " + title + ", Author: " + author);
    }
}
// Level 2: Child class
class Book extends LibraryItem {
    private int pageCount;
    public Book(String title, String author, int pageCount) {
        super(title, author);
        this.pageCount = pageCount;
    }
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Pages: " + pageCount);
    }
}
// Level 3: Grandchild class
class EBook extends Book {
    private String fileFormat;
    public EBook(String title, String author, int pageCount, String fileFormat) {
        super(title, author, pageCount);
        this.fileFormat = fileFormat;
    }
@Override
    public void displayInfo() {
        super.displayInfo();
```

```
        System.out.println("File Format: " + fileFormat);
    }
}
```

2. Abstract Class

An abstract class is like a partially built library: it has some concrete things (like walls) but also some incomplete areas (like shelves) for future development. You can't create an instance of an abstract class directly but can extend it to make complete classes.

Example:

java

```
// Abstract class
abstract class LibraryItem {
    protected String title;
    protected String author;
// Concrete method
    public void displayInfo() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
    }
// Abstract method (to be implemented by child classes)
    public abstract void checkOut();
}
// Concrete class
class Book extends LibraryItem {
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }
@Override
    public void checkOut() {
        System.out.println("Checking out book: " + title);
```

```
    }
}
```

Layman Analogy:

Think of an abstract class as a blueprint for a house with some rooms pre-built but others left for you to decide (kitchen layout, bedroom design). The builder (child class) will complete it.

3. Interfaces

An interface is like a library policy: it defines rules that all types of items (books, magazines, DVDs) must follow, but how they follow these rules is up to them. Classes can implement multiple interfaces to follow multiple policies.

Example:

java

```
// Interface for borrowable items
interface Borrowable {
    void checkOut();
    void returnItem();
}
// Interface for reservable items
interface Reservable {
    void reserve();
}
// Book class implementing both interfaces
class Book implements Borrowable, Reservable {
    private String title;
    public Book(String title) {
        this.title = title;
    }
@Override
    public void checkOut() {
        System.out.println("Checking out book: " + title);
    }
@Override
```

```
    public void returnItem() {
        System.out.println("Returning book: " + title);
    }
@Override
    public void reserve() {
        System.out.println("Reserving book: " + title);
    }
}
```

Layman Analogy:

Imagine interfaces as a contract with a rulebook (like a librarian setting a rule for borrowing books). Any item (book, magazine) must follow the rules but can implement the borrowing or reserving in its own way.

Summary:

- Inheritance: Allows child classes to inherit methods/fields from parent classes.
- Abstract Class: Defines common behavior but expects subclasses to provide the specific implementation.
- Interface: Sets rules that implementing classes must follow, but how they follow the rules is up to the class.

These concepts are fundamental in object-oriented programming and provide flexibility when designing systems like a Library Management System.

Follow Us On Our Social Media Platforms:

Instagram: https://www.instagram.com/algorithms365/

YouTube: https://www.youtube.com/@algorithms365

WhatsApp: https://chat.whatsapp.com/K1K7wDMEXG0DJhqMCxFtht

LinkedIn: https://in.linkedin.com/company/algorithms365

Facebook: https://www.facebook.com/algorithms365

Web Site: https://algorithms365.com/