Java OOP Concepts Demonstrated: Library Management System

Overview :

This project demonstrates key **Java Object-Oriented Programming (OOP)** concepts through a simple **Library Management System**:

- **Abstraction**: Using abstract classes to define common behaviour.
- **Inheritance**: Single and multilevel inheritance to reuse code.
- **Polymorphism**: Method overloading and overriding for dynamic behaviour.
- **Interfaces**: For contract-based design.
- **Encapsulation**: Using private/protected access modifiers and getters/setters.

1. Abstract Class: LibraryItem.java

```
// Represents a generic library item
public abstract class LibraryItem {
    protected String title;
    protected String author;

    // Constructor
    public LibraryItem(String title, String author) {
        this.title = title;
        this.author = author;
    }

    // Abstract method to print details of the library item
    public abstract void printDetails();

    // Optional getters for encapsulation
    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }
}
```

2. Inheritance and Method Overriding: Book.java

```
// Represents a physical book
public class Book extends LibraryItem {
    private String genre;

    public Book(String title, String author, String genre) {
        super(title, author); // Call parent constructor
        this.genre = genre;
    }

    @Override
    public void printDetails() {
        System.out.println("Book: " + title + ", Author: " + author + ", Genre: " + genre);
    }

    // Getter for genre
    public String getGenre() {
        return genre;
    }

    // Method overloading example
    public void printDetails(boolean showGenre) {
        if (showGenre) {
            printDetails();
        } else {
            System.out.println("Book: " + title + ", Author: " + author);
        }
    }
}
```

3. Interface: Borrowable.java

```
// Interface defining borrowable behaviour
public interface Borrowable {
    void borrowItem(String borrower);
    void returnItem();
}
```

4. Multilevel Inheritance and Interface Implementation: Ebook.java

```
// Represents an eBook that extends Book and is borrowable
public class Ebook extends Book implements Borrowable {
    private boolean isBorrowed;

    public Ebook(String title, String author, String genre) {
        super(title, author, genre);
        this.isBorrowed = false;
    }

    @Override
    public void borrowItem(String borrower) {
        if (!isBorrowed) {
            isBorrowed = true;
            System.out.println(borrower + " borrowed the eBook: " + getTitle());
        } else {
            System.out.println("eBook already borrowed: " + getTitle());
        }
    }

    @Override
    public void returnItem() {
        if (isBorrowed) {
            isBorrowed = false;
            System.out.println("eBook returned: " + getTitle());
        } else {
            System.out.println("eBook was not borrowed: " + getTitle());
        }
    }

    // Getter to check borrowing status
    public boolean isBorrowed() {
        return isBorrowed;
    }
}
```

5. Application Demo: Main.java

```
public class Application {
    public static void main(String[] args) {
        // Creating a book object
        Book book = new Book("Java Basics", "John Doe", "Programming");

        // Creating an eBook object
        Ebook ebook = new Ebook("Effective Java", "Joshua Bloch", "Tech");

        // Demonstrate method overriding
        book.printDetails();
        ebook.printDetails();

        // Demonstrate method overloading
        book.printDetails(false); // Print without genre

        // Demonstrate interface methods
        ebook.borrowItem("Veeresh");
        ebook.borrowItem("Alex"); // Attempt to borrow already borrowed book
        ebook.returnItem();
        ebook.returnItem(); // Attempt to return again
    }
}
```