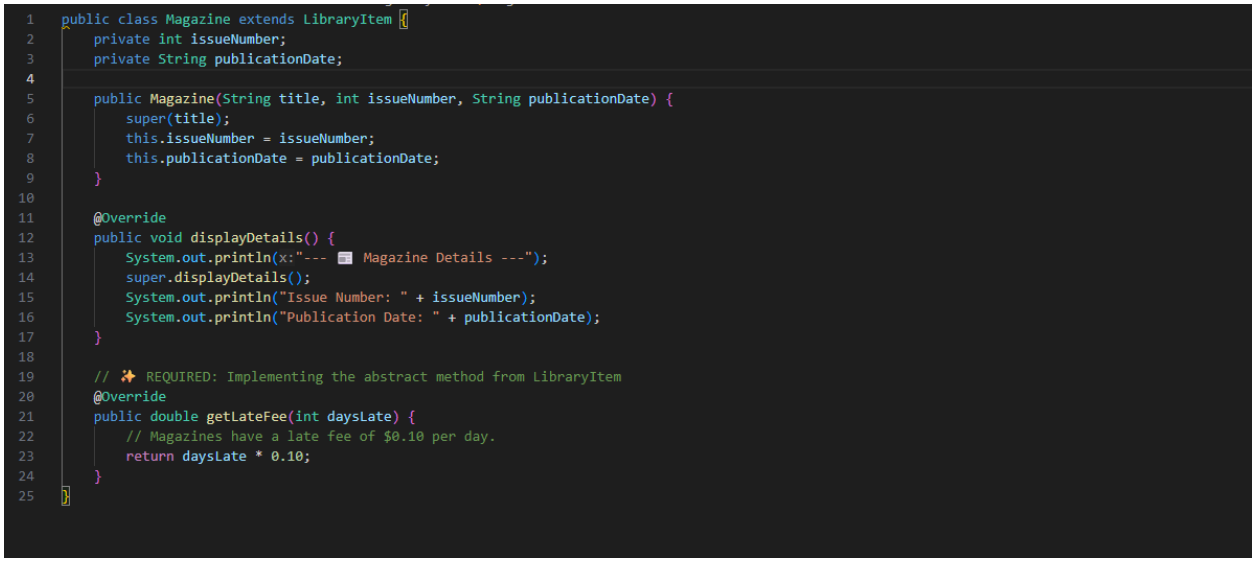
```java
import java.util.UUID;

// Now an abstract class to serve as a true blueprint
public abstract class LibraryItem {
    protected String title;
    protected String itemId;
    protected boolean isCheckedOut;

    public LibraryItem(String title) {
        this.title = title;
        this.itemId = UUID.randomUUID().toString().substring(0, 8);
        this.isCheckedOut = false;
    }

    // ✨ NEW: An abstract method.
    // Subclasses MUST provide their own implementation for this method.
    public abstract double getLateFee(int daysLate);

    // --- DEMONSTRATING METHOD OVERLOADING ---
    // Method 1: No parameters
    public void checkOut() {
        this.isCheckedOut = true;
        System.out.println("'" + title + "' has been checked out.");
    }

    // Method 2: Same name, different parameter (overloaded)
    public void checkOut(String dueDate) {
        this.isCheckedOut = true;
        System.out.println("'" + title + "' has been checked out. Please return by " + dueDate + ".");
    }
    // ----------------------------------------

    public void returnItem() {
        this.isCheckedOut = false;
        System.out.println("'" + title + "' has been returned.");
    }

    public void displayDetails() {
        System.out.println("Title: " + title);
        System.out.println("Item ID: " + itemId);
        System.out.println("Status: " + (isCheckedOut ? "Checked Out" : "Available"));
    }
}
```

```java
public class Book extends LibraryItem {
    private String author;
    private String isbn;

    public Book(String title, String author, String isbn) {
        super(title);
        this.author = author;
        this.isbn = isbn;
    }

    // Overriding the displayDetails method (already done)
    @Override
    public void displayDetails() {
        System.out.println("--- 📖 Book Details ---");
        super.displayDetails();
        System.out.println("Author: " + author);
        System.out.println("ISBN: " + isbn);
    }

    // ✨ REQUIRED: Implementing the abstract method from LibraryItem
    @Override
    public double getLateFee(int daysLate) {
        // Books have a late fee of $0.25 per day.
        return daysLate * 0.25;
    }
}
```



```java
public class Magazine extends LibraryItem {
    private int issueNumber;
    private String publicationDate;

    public Magazine(String title, int issueNumber, String publicationDate) {
        super(title);
        this.issueNumber = issueNumber;
        this.publicationDate = publicationDate;
    }

    @Override
    public void displayDetails() {
        System.out.println(x:"--- 📰 Magazine Details ---");
        super.displayDetails();
        System.out.println("Issue Number: " + issueNumber);
        System.out.println("Publication Date: " + publicationDate);
    }

    // ✨ REQUIRED: Implementing the abstract method from LibraryItem
    @Override
    public double getLateFee(int daysLate) {
        // Magazines have a late fee of $0.10 per day.
        return daysLate * 0.10;
    }
}
```

```java
public class DVD extends LibraryItem {
    private String director;
    private int runtimeMinutes;

    public DVD(String title, String director, int runtimeMinutes) {
        super(title);
        this.director = director;
        this.runtimeMinutes = runtimeMinutes;
    }

    @Override
    public void displayDetails() {
        System.out.println(x:"--- 💿 DVD Details ---");
        super.displayDetails();
        System.out.println("Director: " + director);
        System.out.println("Runtime: " + runtimeMinutes + " minutes");
    }

    // ✨ REQUIRED: Implementing the abstract method from LibraryItem
    @Override
    public double getLateFee(int daysLate) {
        // DVDs have a higher late fee of $1.00 per day.
        return daysLate * 1.00;
    }
}
```

```java
public class ProjectTester {
    Run | Debug
    public static void main(String[] args) {
        System.out.println(x:"============== EXER4: POLYMORPHISM TEST ==============\n");

        // 1. ✨ DEMONSTRATING RUNTIME POLYMORPHISM (Method Overriding) ✨
        // We create an array of the SUPERCLASS type (LibraryItem).
        // But we fill it with objects of the SUBCLASS types.
        // This is polymorphism in action!

        LibraryItem[] libraryItems = new LibraryItem[3];
        libraryItems[0] = new Book(title:"The Lord of the Rings", author:"J.R.R. Tolkien", isbn:"978-0618640157");
        libraryItems[1] = new Magazine(title:"TIME Magazine", issueNumber:102, publicationDate:"September 2025");
        libraryItems[2] = new DVD(title:"The Matrix", director:"Wachowskis", runtimeMinutes:136);

        System.out.println(x:"--- Iterating through Library Items and calling common methods ---");

        // Loop through the array. Even though we call 'item.displayDetails()',
        // Java knows to run the specific version of the method for each object (Book, Magazine, or DVD).
        for (LibraryItem item : libraryItems) {
            item.displayDetails();
            // We can also call the new abstract method we implemented.
            int daysLate = 5;
            double fee = item.getLateFee(daysLate);
            System.out.printf(format:"Late fee for %d days: $%.2f%n", daysLate, fee);
            System.out.println(x:"-----------------------------------");
        }

        System.out.println(x:"\n\n============== END OF RUNTIME POLYMORPHISM TEST ==============\n\n");

        // 2. ✨ DEMONSTRATING COMPILE-TIME POLYMORPHISM (Method Overloading) ✨
        System.out.println(x:"--- Testing Overloaded 'checkOut' Method on a Book object ---");

        Book anotherBook = new Book(title:"Dune", author:"Frank Herbert", isbn:"978-0441013593");

        // Calling the first version of checkOut() (no parameters)
        anotherBook.checkOut();

        // Calling the second, overloaded version of checkOut() (with a String parameter)
        anotherBook.checkOut(dueDate:"October 23, 2025");

        System.out.println(x:"\n============== END OF COMPILE-TIME POLYMORPHISM TEST ==============");
    }
}
```