Day 21 - 104608492 - Shirisha Perapagu

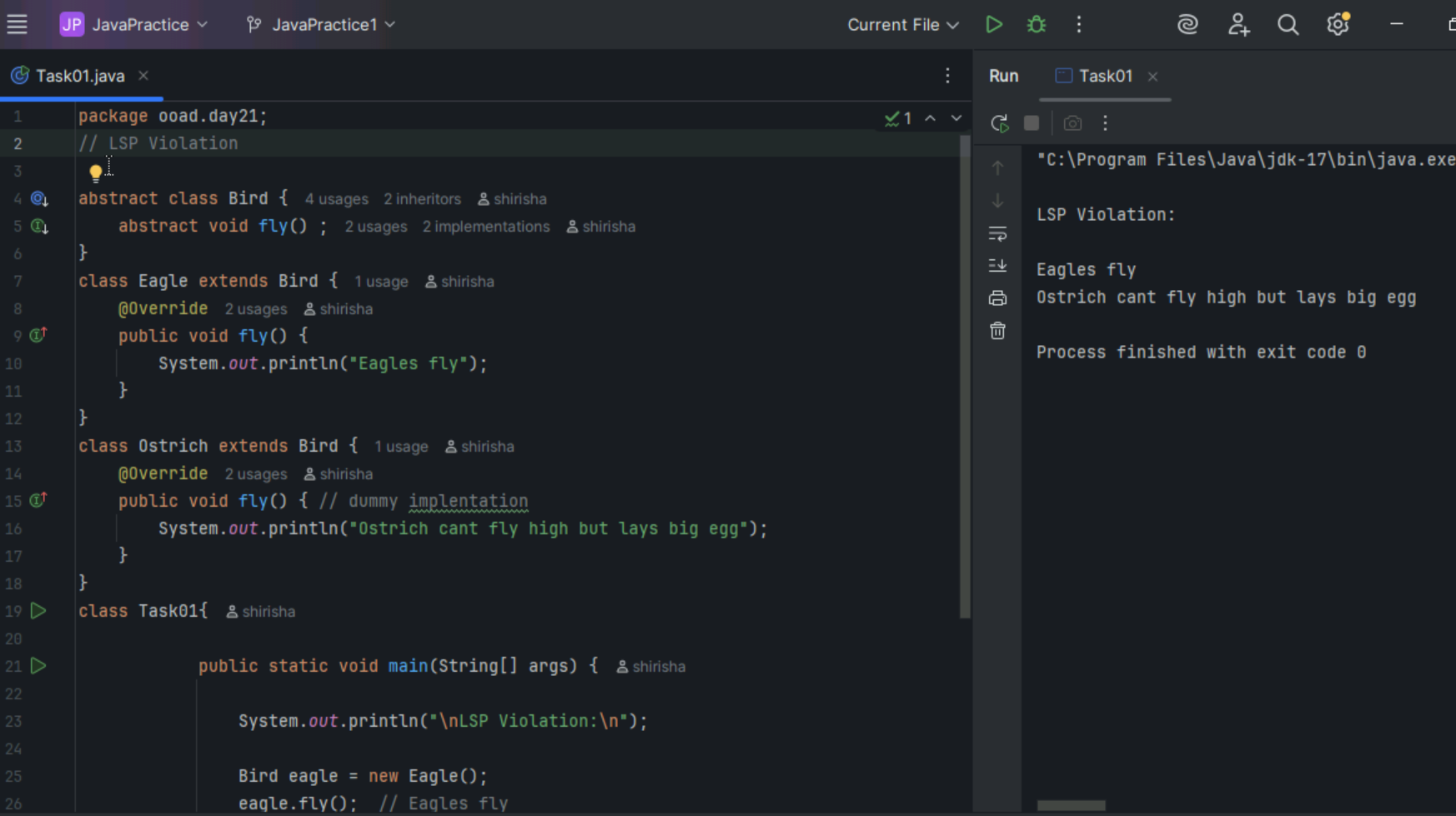
**SOLID Principles Continuation….**

L - Liskov Substitution Principle

Task01

Violation of LSP

Ostrich class extends Bird class but logically cannot fly but still it is forced to implement fly().



Task 02

LSP Implementation

Here we are separating birds that fly and don’t fly in separate abstract classes, so that birds that fly can only implement fly().



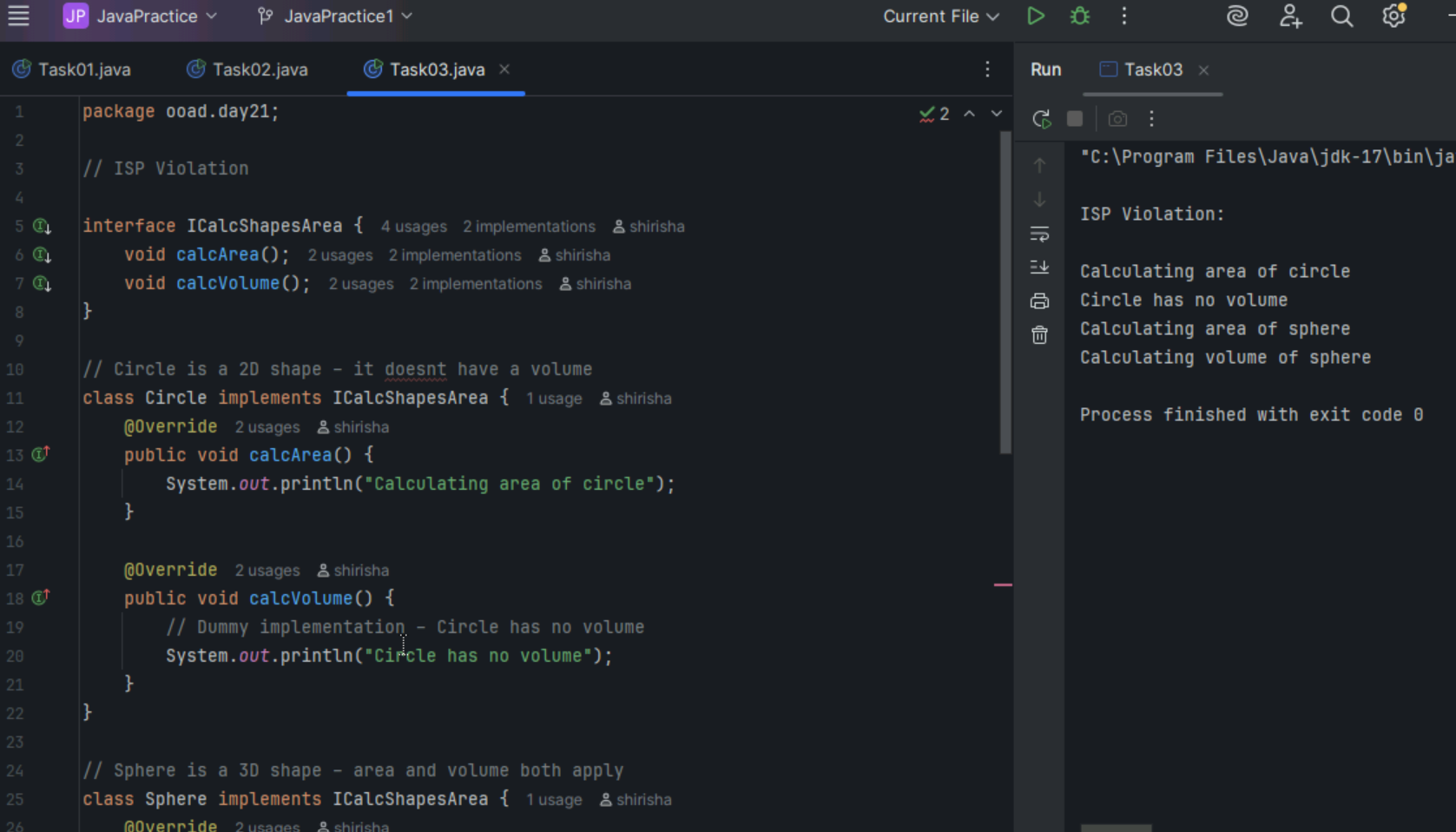
I -

ISP - Interface Segregation Principle

Task 03

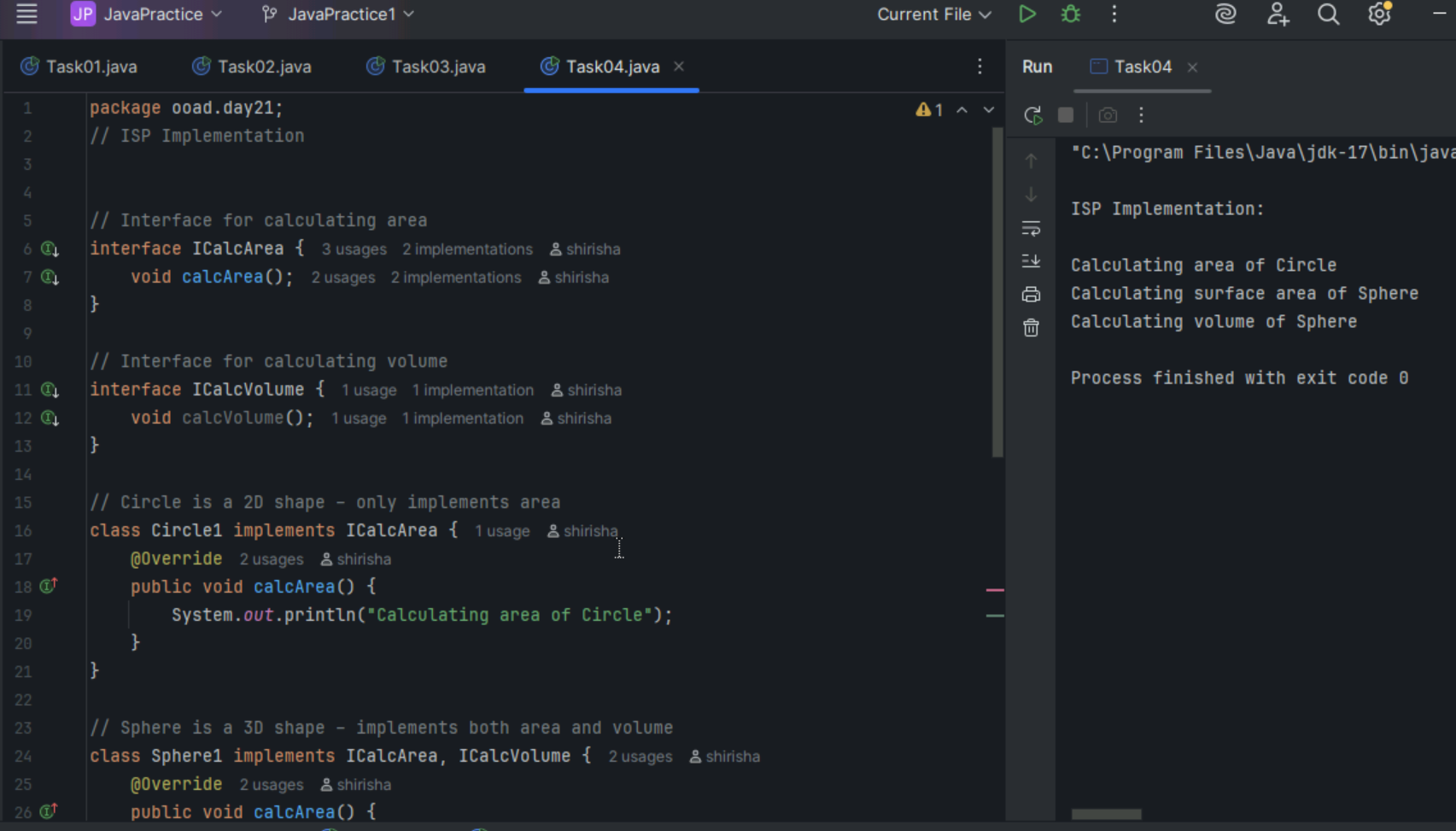
ISP Violation

Here Circle class is forced to implement calcVolume() even though circle doesn’t have a volume.



Task 04

ISP Implementation – separate interface for calculating volume so that Circle need not implement it.



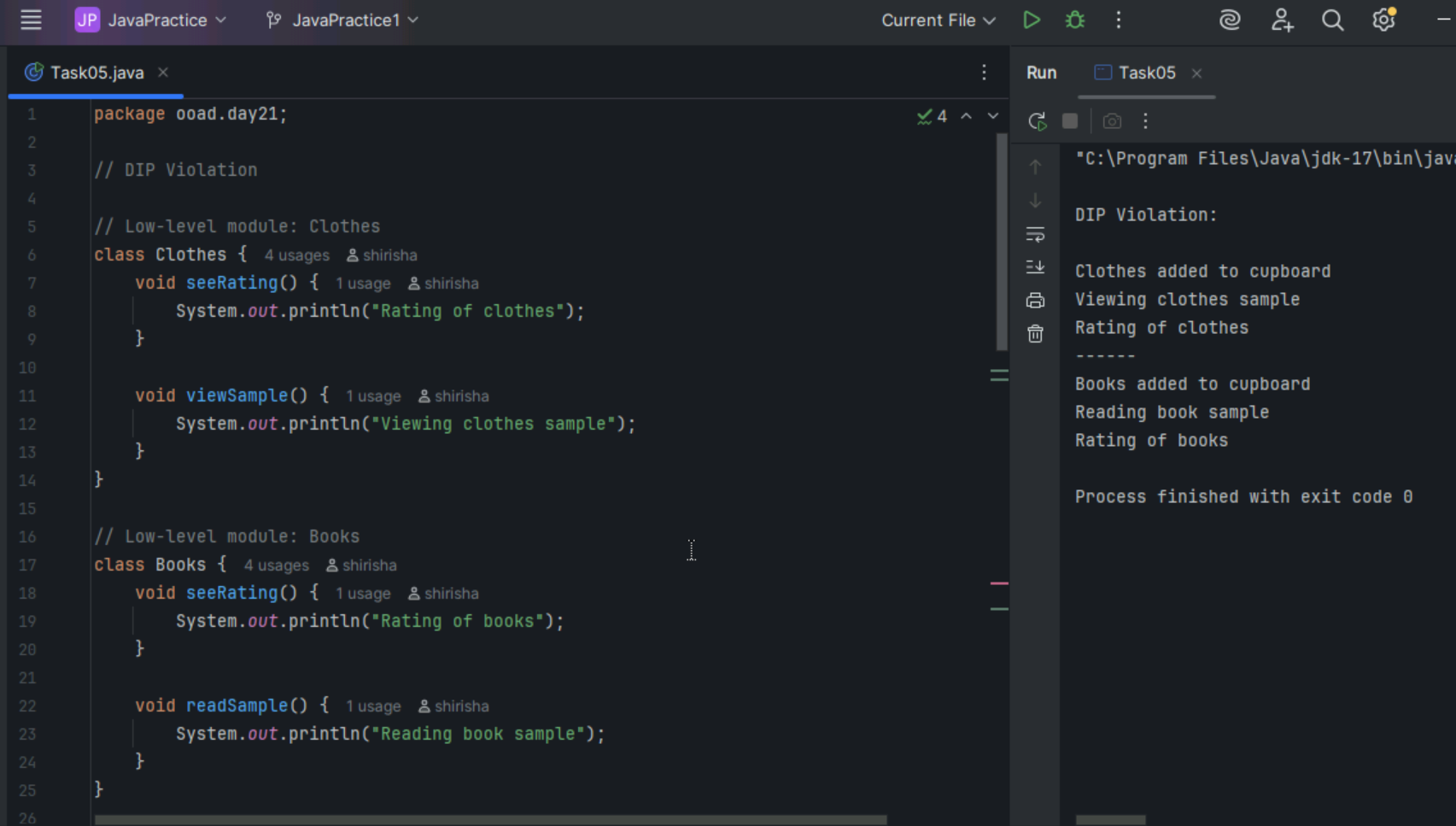
D -

DIP - Dependency Inversion Principle

Task 05

DIP Violation

Here Cupboard class(High-level module) directly depends on concrete classes Clothes, Books(Low-level modules) instead of abstracting over them. So if we want to add new items like Vessels then we have to change Cupboard violating DIP.

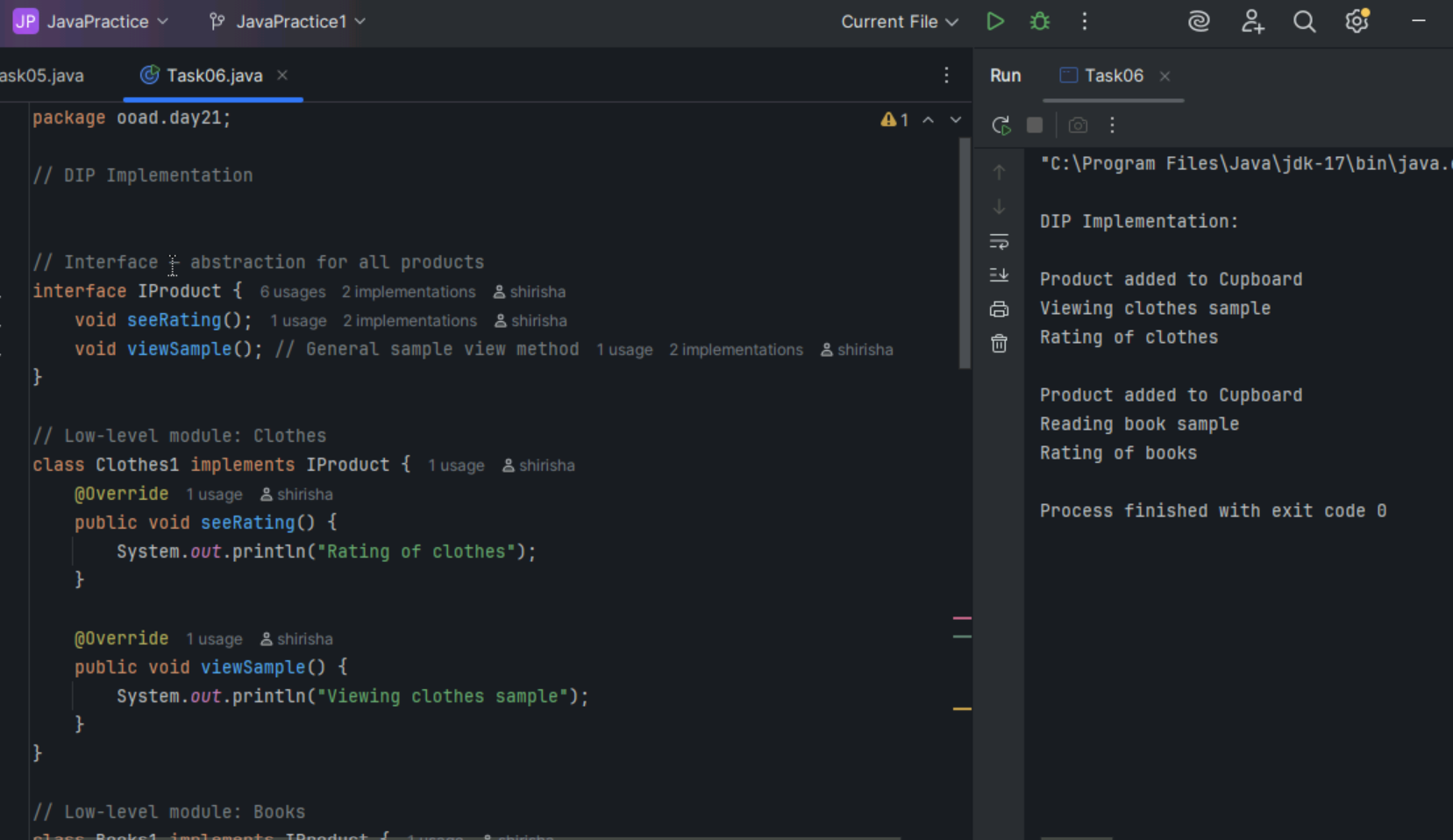


Task 06

DIP Implementation

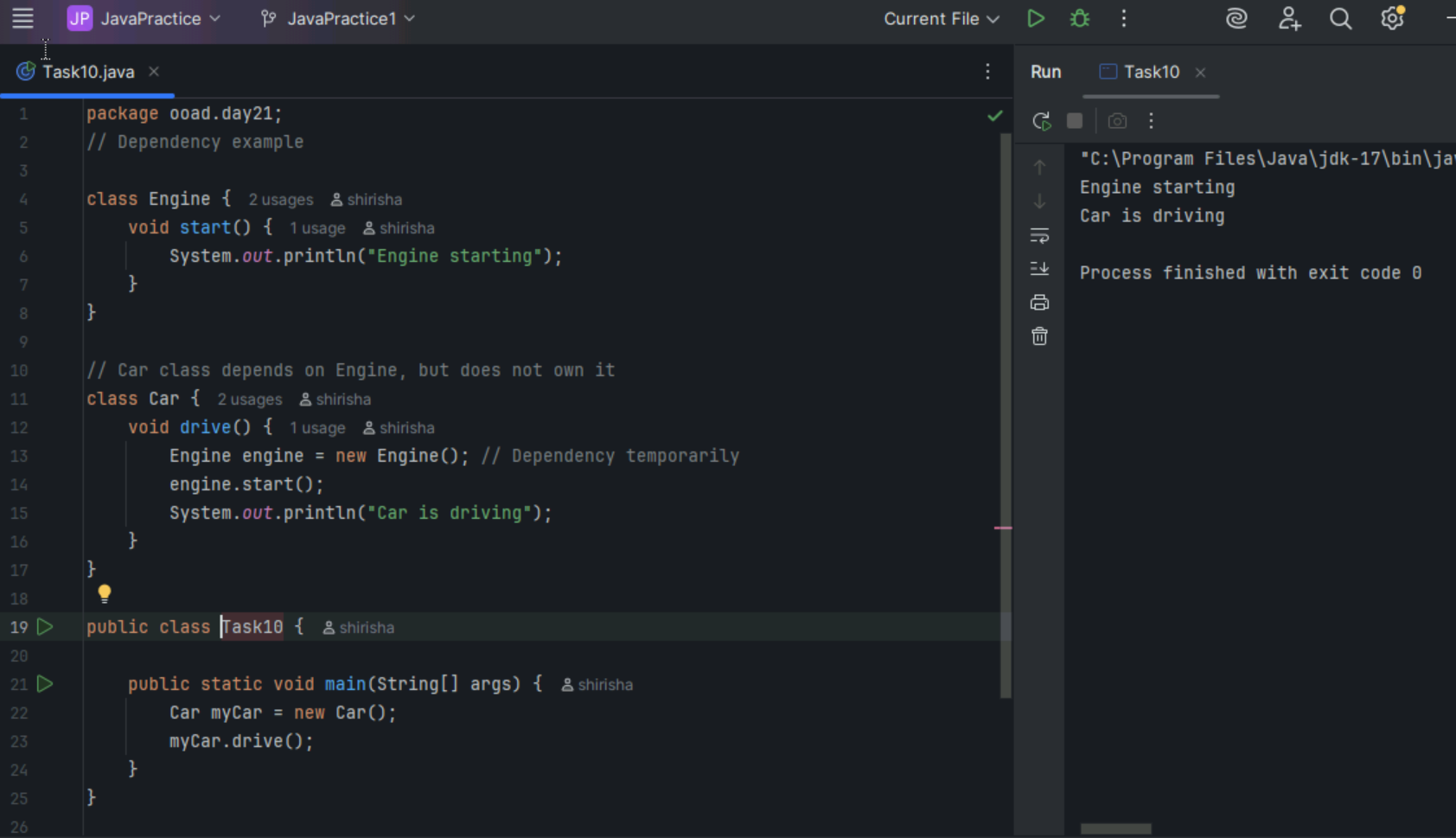
Cupboard1 does not depend on specific classes like Clothes1 or Books1.

It only depends on the IProduct interface, allowing to plug in any product type in future without changing Cupboard1.



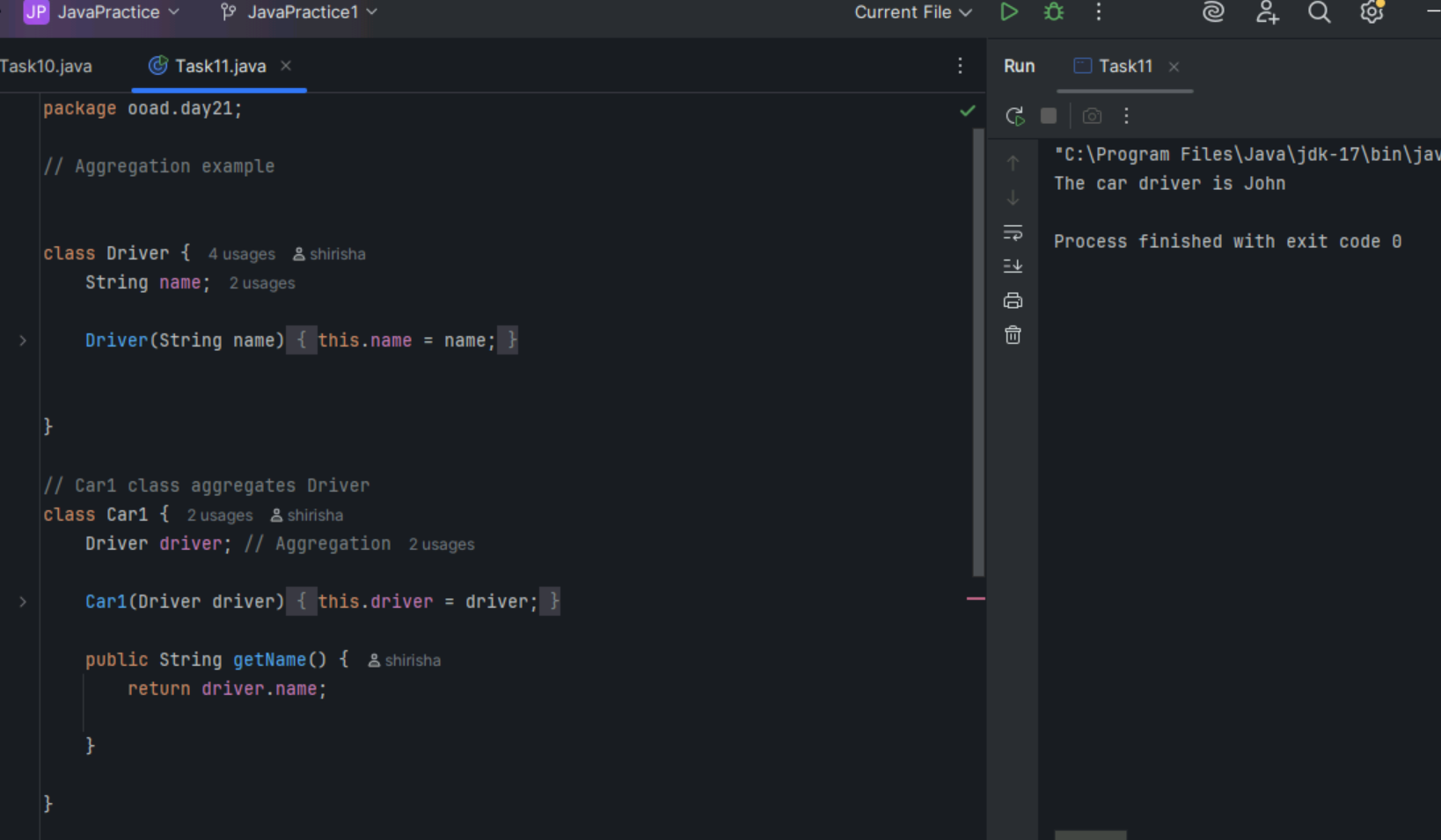
Task 10 – 12 (From Doc 07)

Task 10 Dependency example

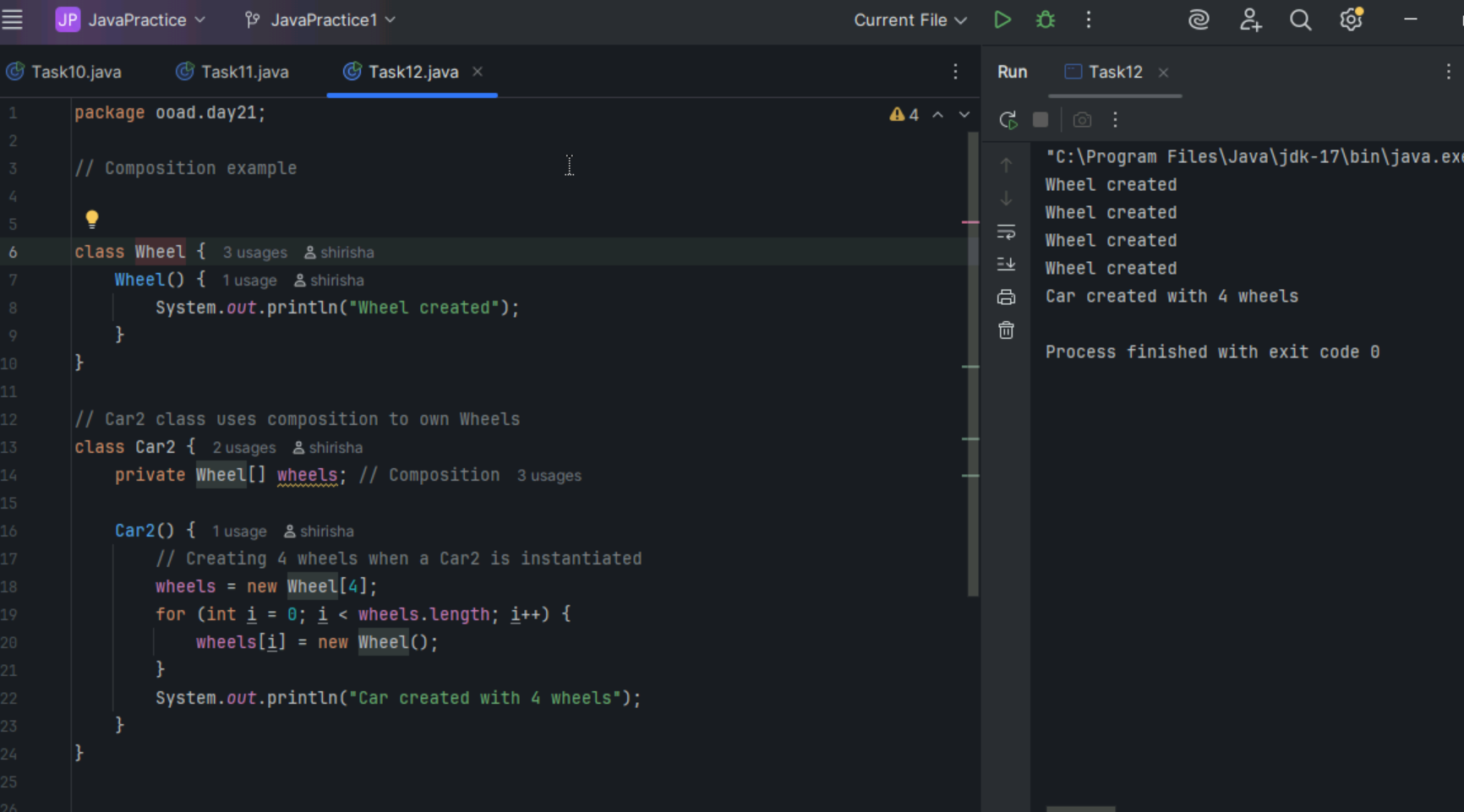


Task 11

Aggregation example



Task 12 Composition example



Other examples:

**Dependency** - uses temporarily

Example:

Person borrowing a Pen temporarily.

Person depends on pen but does not own it.

**Aggregation** - has but can exist independently

Example:

A School has Students but students can exist without the school.

**Composition** - owns and can’t exist separately

Example:

House having Rooms.

If the house is destroyed, the rooms are gone too.

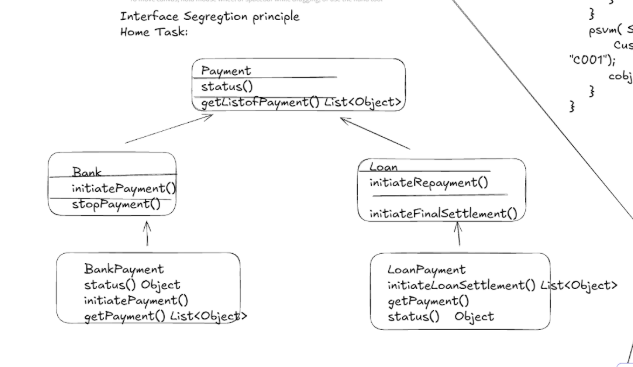
========================================================================

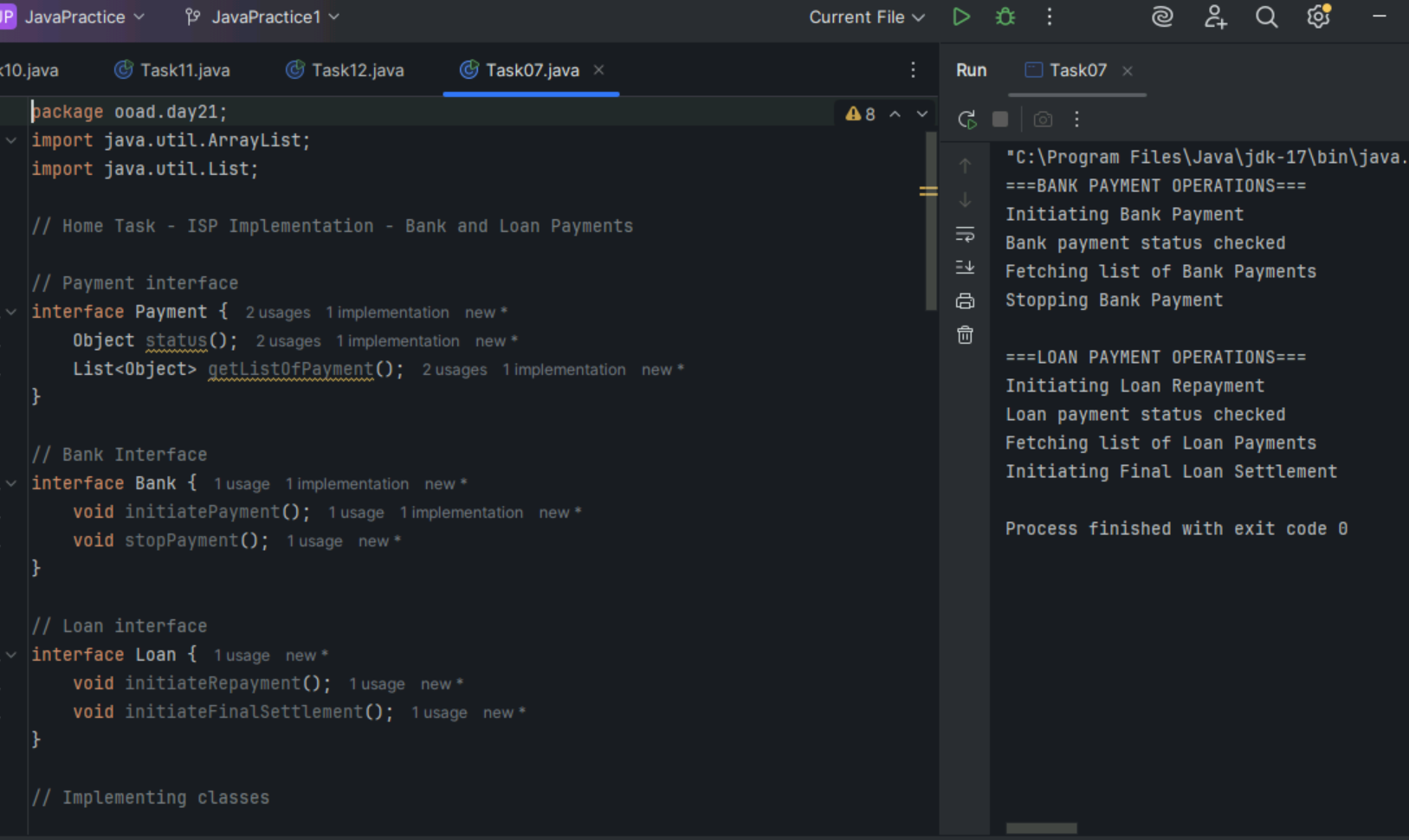
Home Tasks

=========================================================================

Task 07

Can you create a code based on the below diagrammatic representation



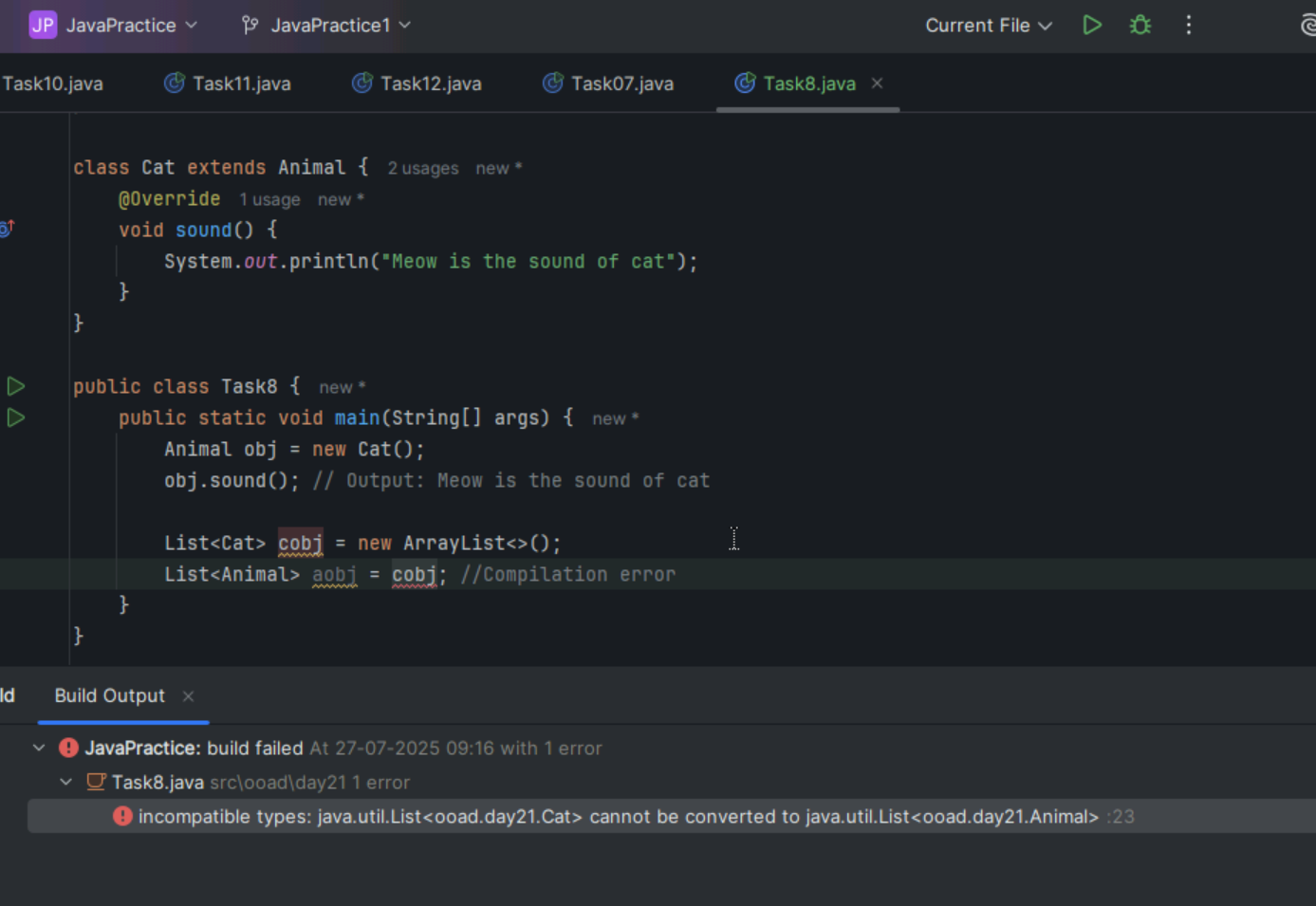


Task 08

Generics Substitution Issue and solution using wildcard

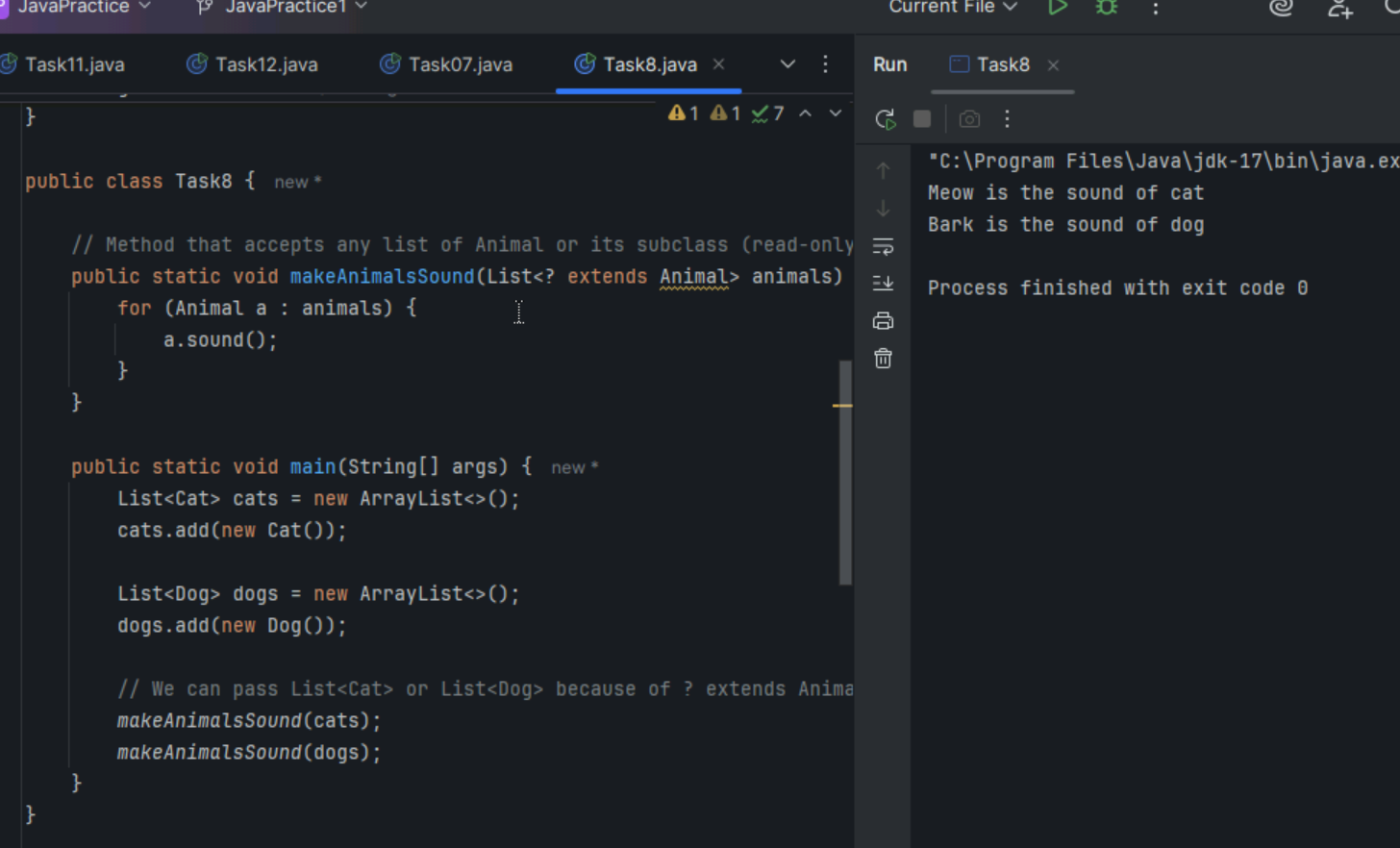
1. Generics Substitution Issue

Even though Cat is a subtype of Animal, List<Cat> is not a subtype of List<Animal> due to invariant generics so directly assigning or passing causes compiler error.



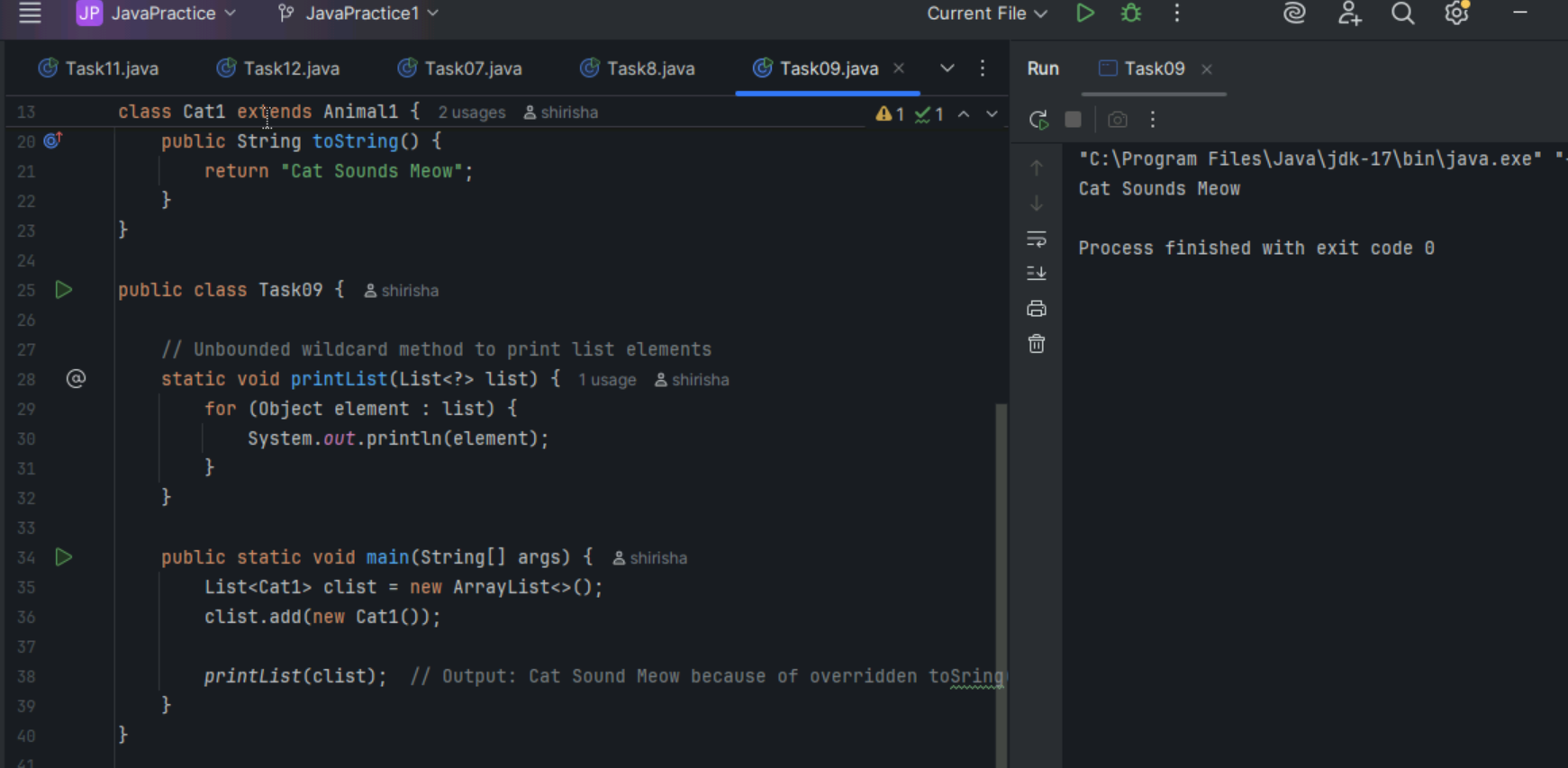
1. Soultion : using wildcards

To avoid Generics Substitution Issue we use wildcard (? extends Animal ) in the method parameter to allow a List of any subclass of Animal like List<Cat> or List<Dog>.



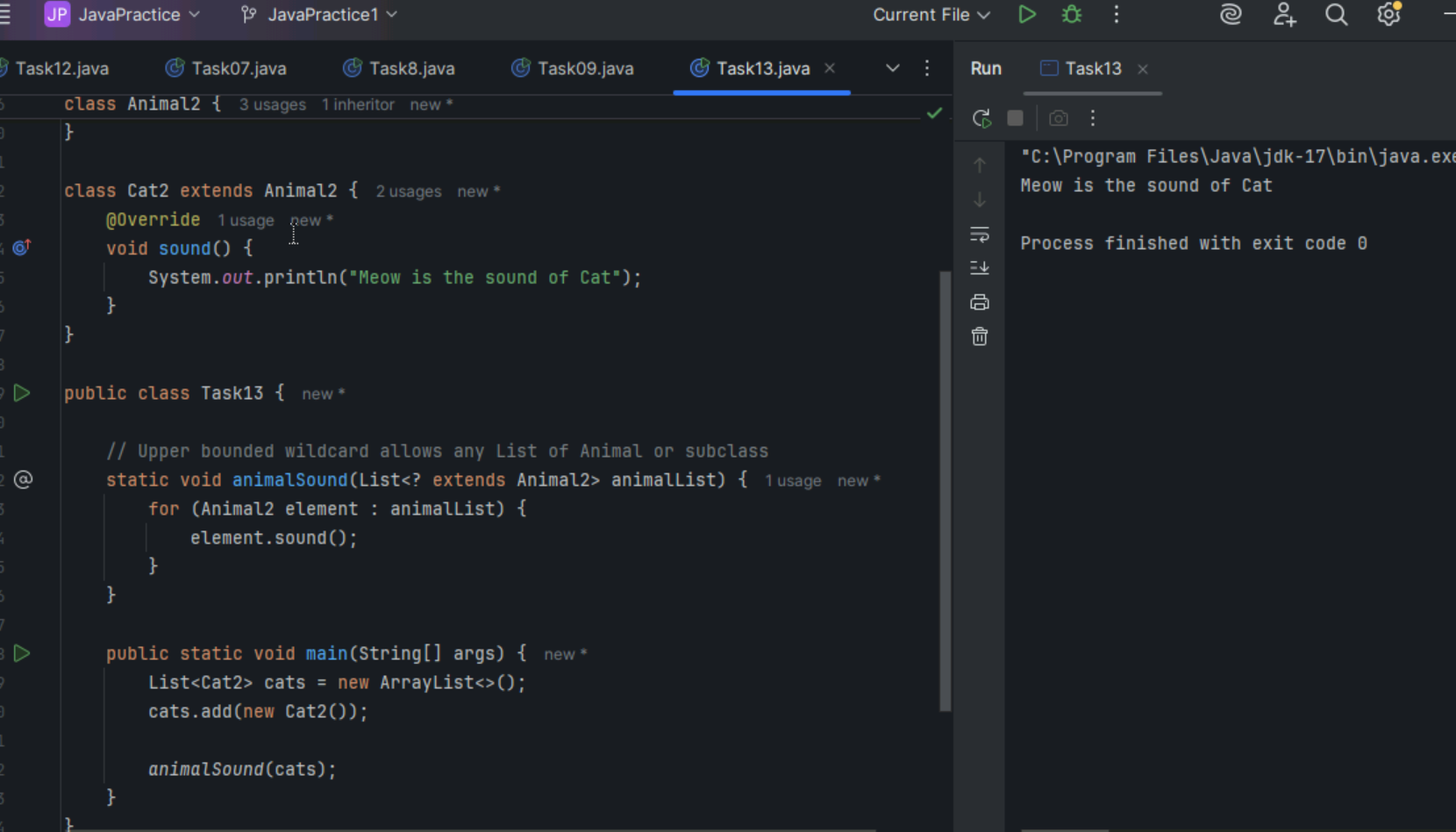
Task 9

Unbounded wildcard

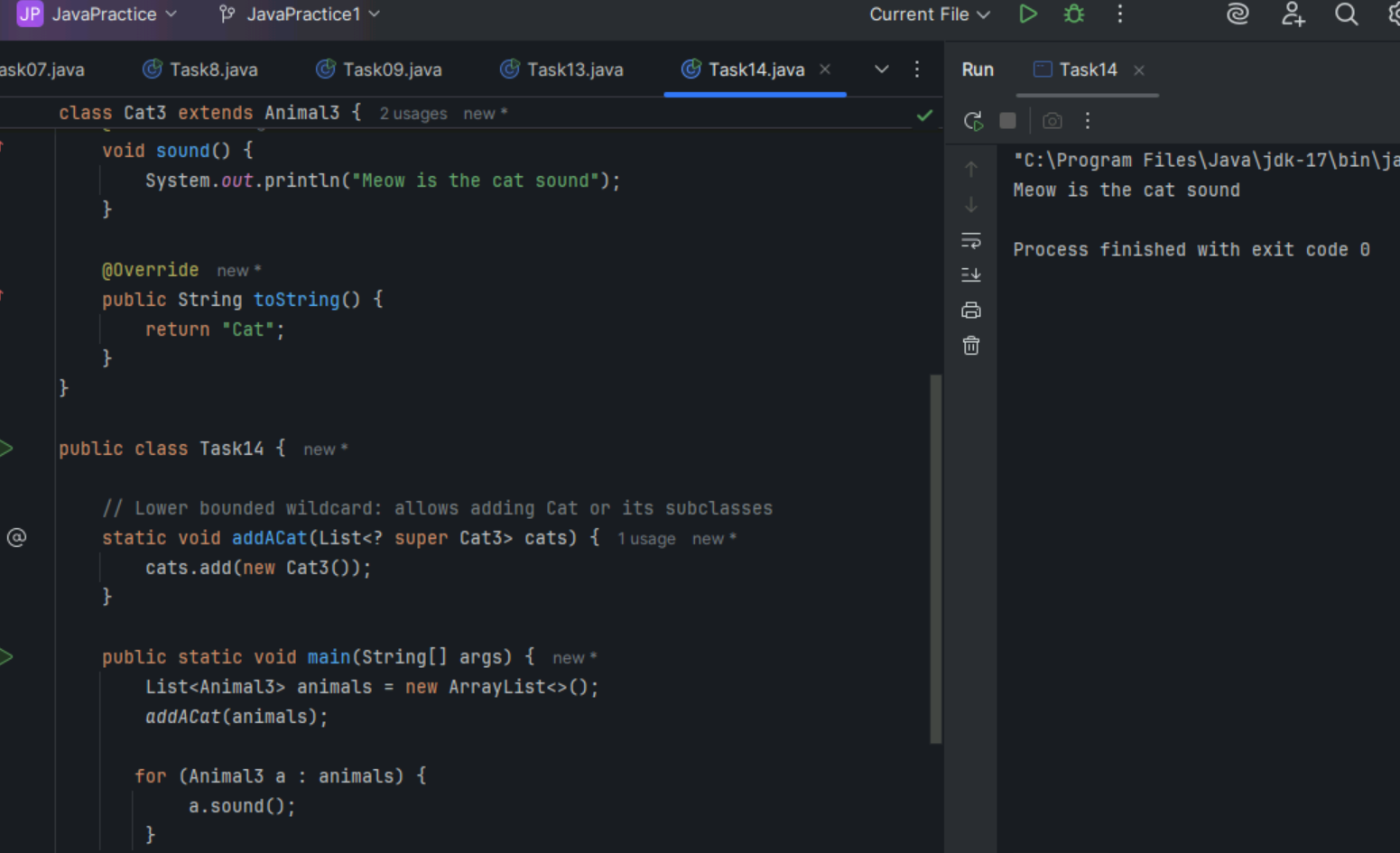


Task10 - Task12 done above HomeTask section.

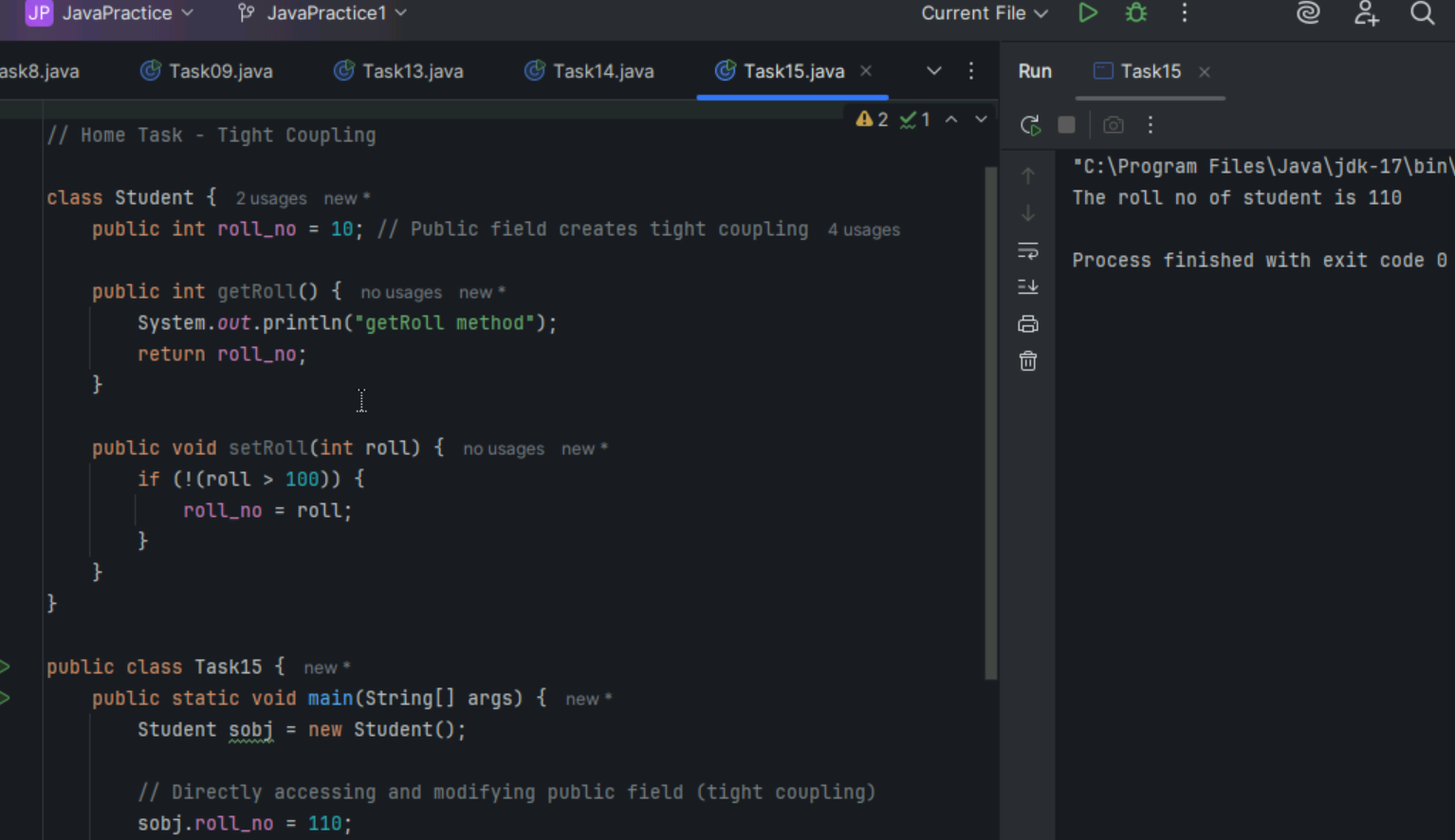
Task 13 : Upper bounded wildcard



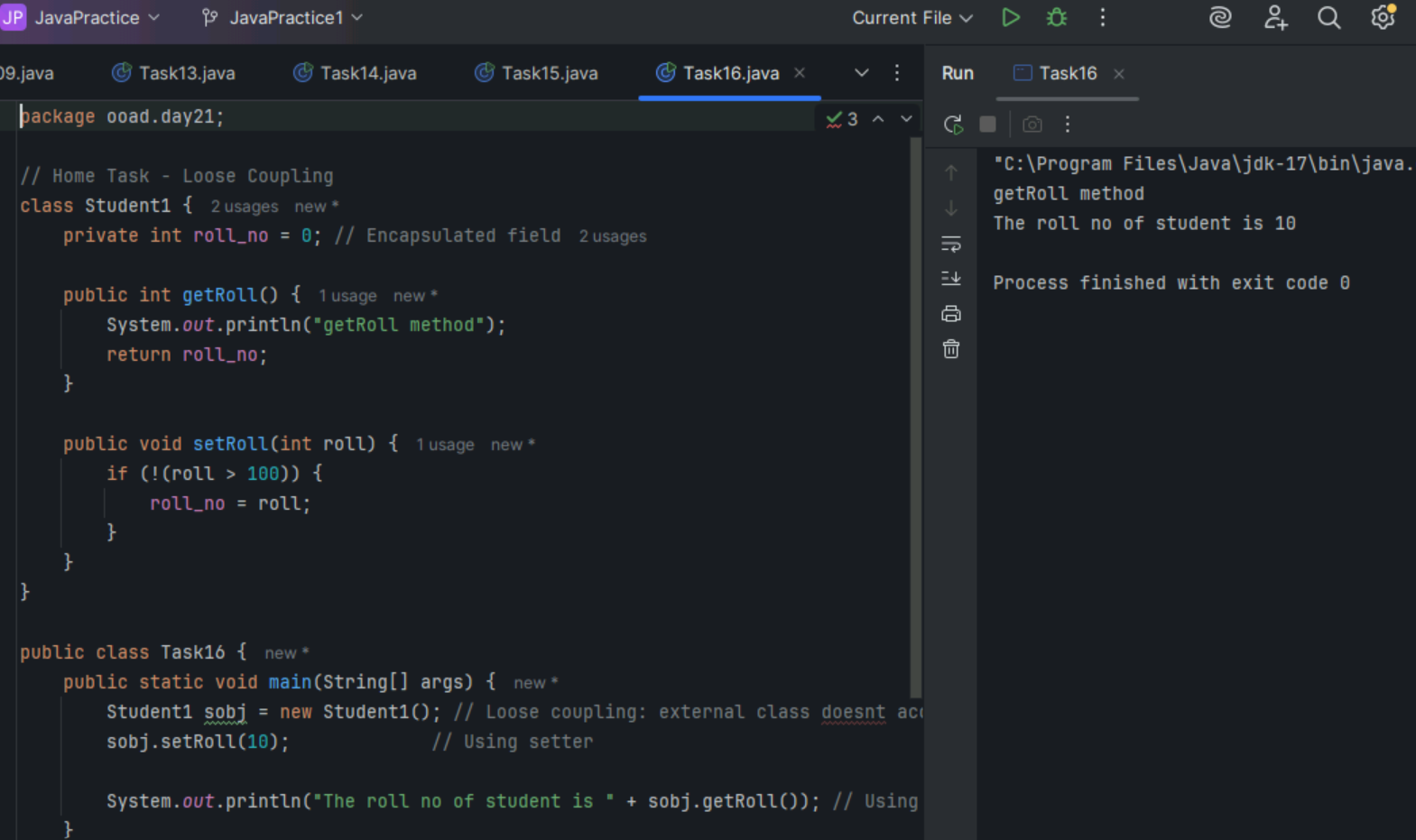
Task 14: Lower bounded wildcard



Task 15: Tight Coupling



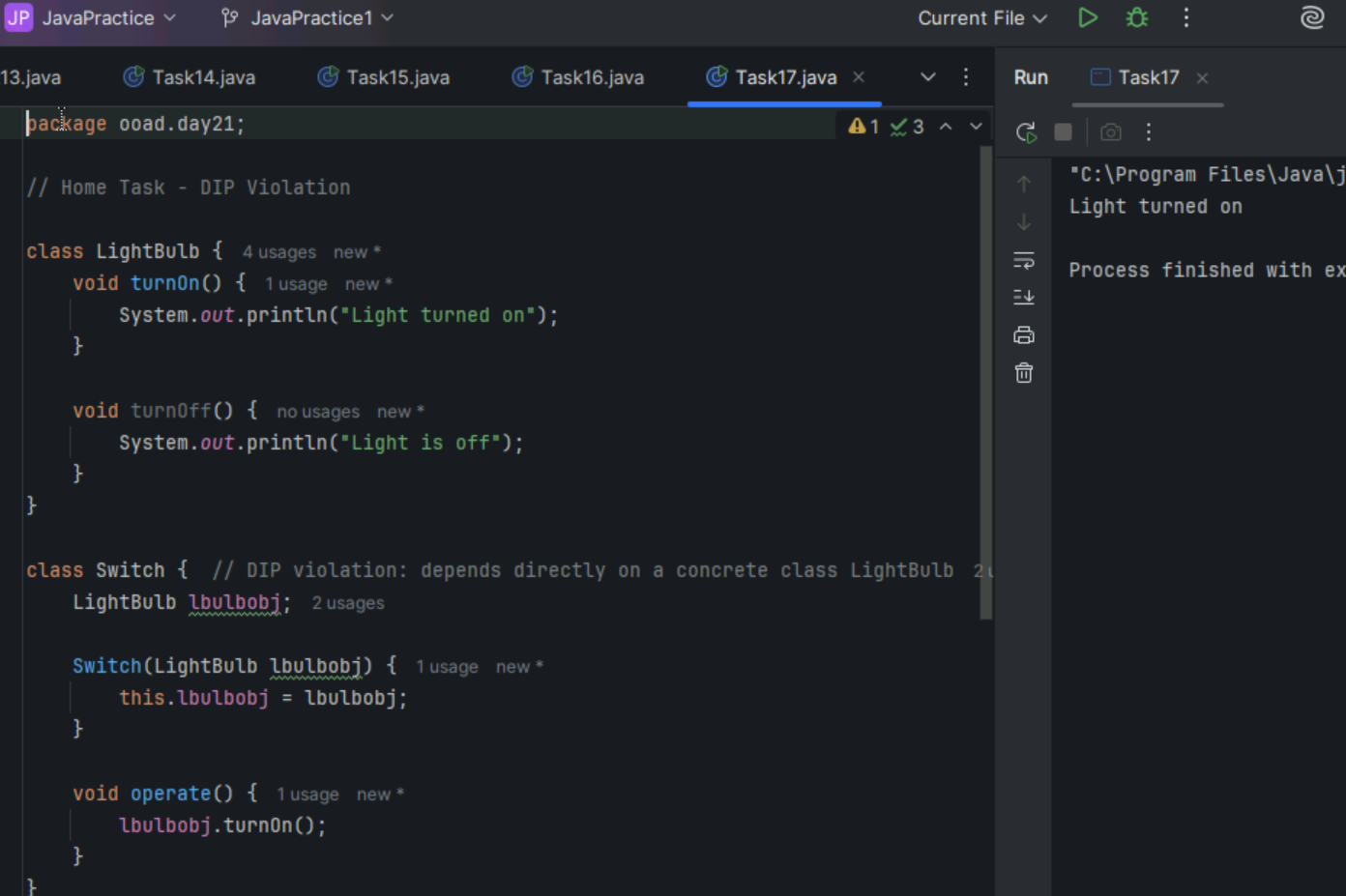
Task 16: Loose Coupling



Task 17

DIP Violation

Switch directly depends on the concrete class LightBulb instead of an abstraction violating DIP.



Task 18

DIP Implementation

Switch class depends on an interface SwitchOnOff that LightBulb class implements, inverting the dependency to an abstraction.

