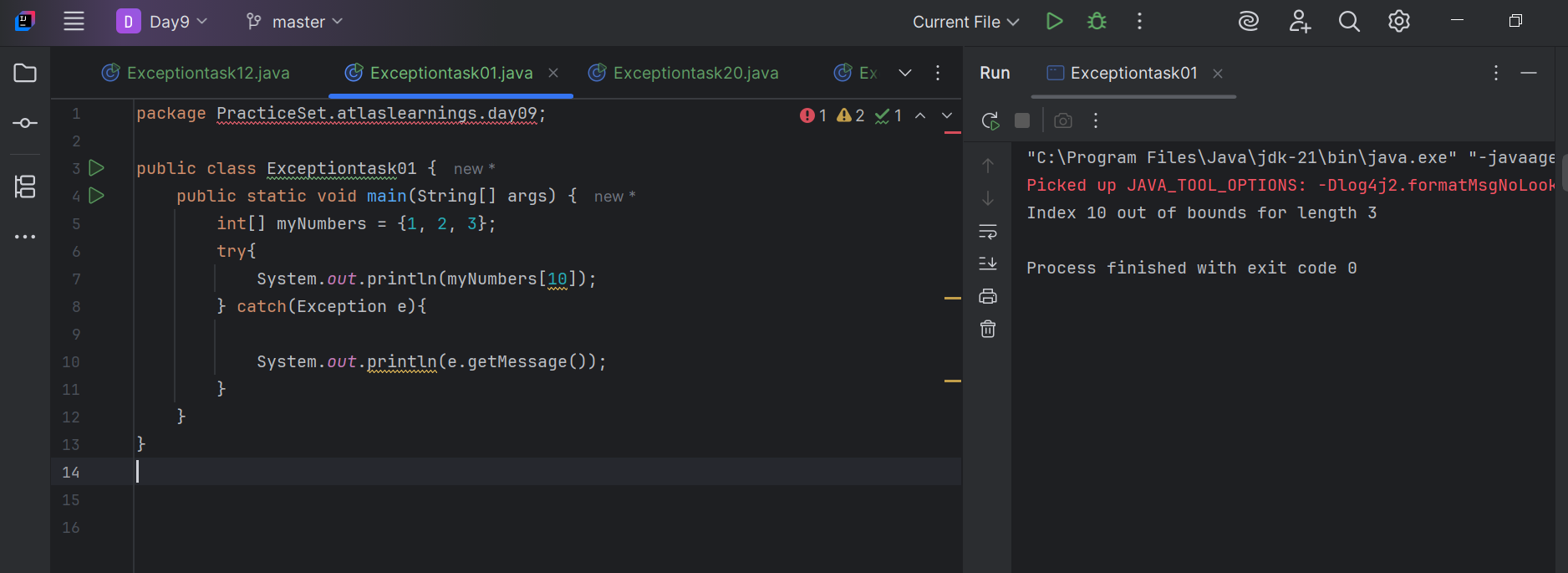
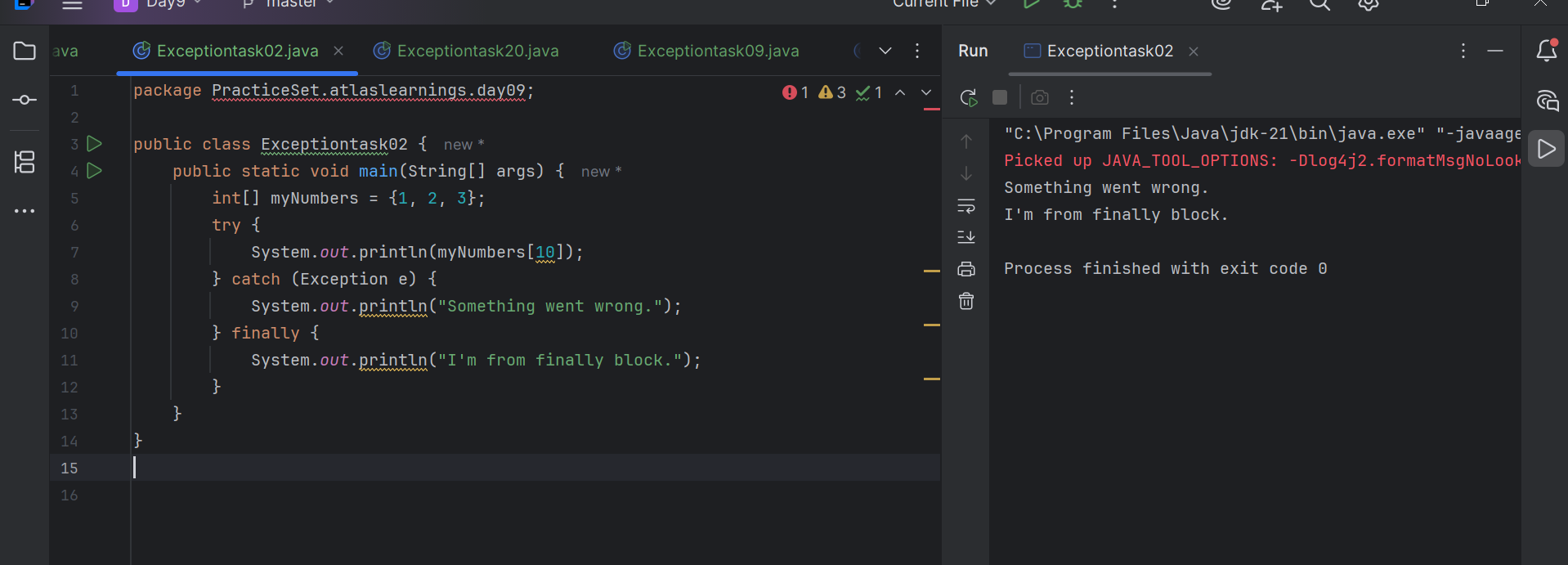
**Day- 9 docs**

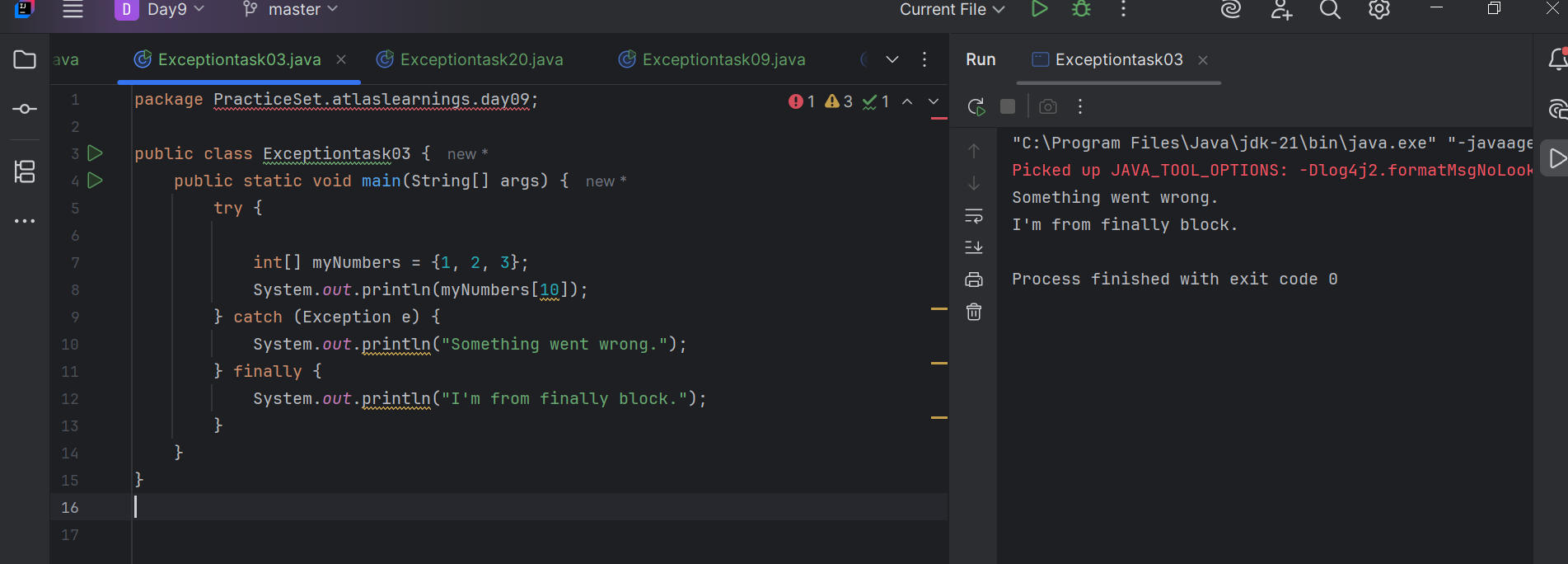
Task -1:



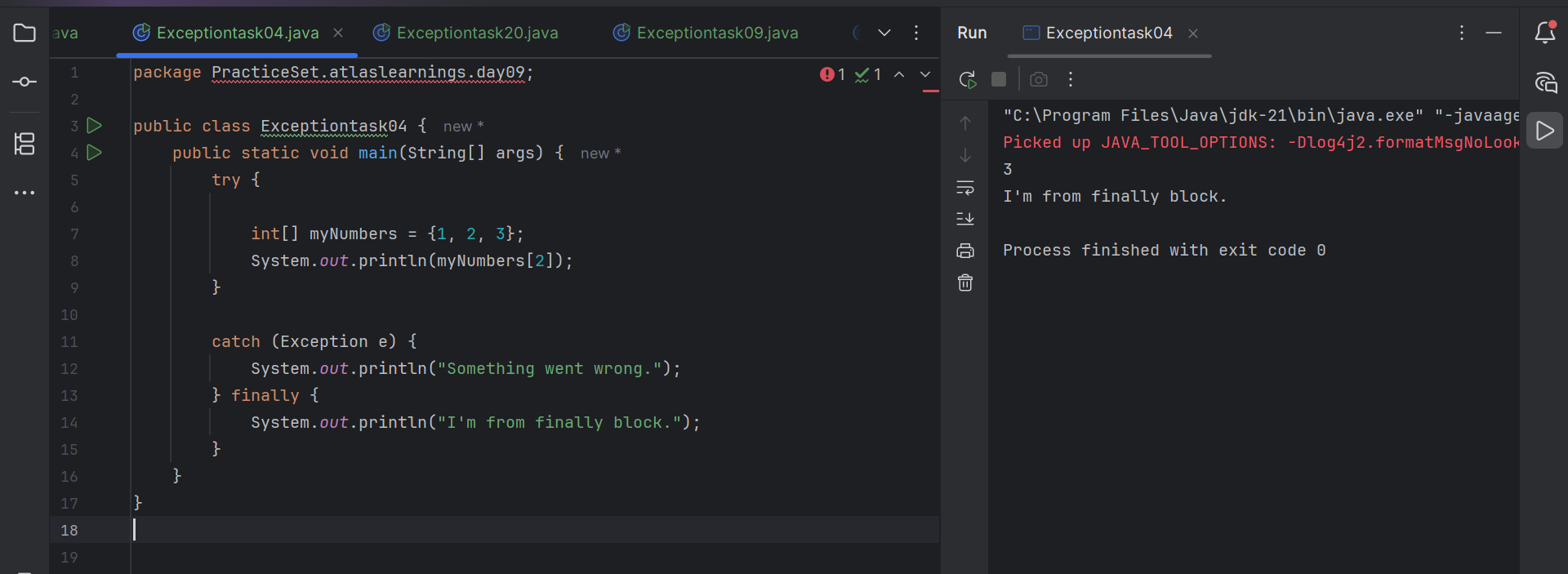
Task - 2:



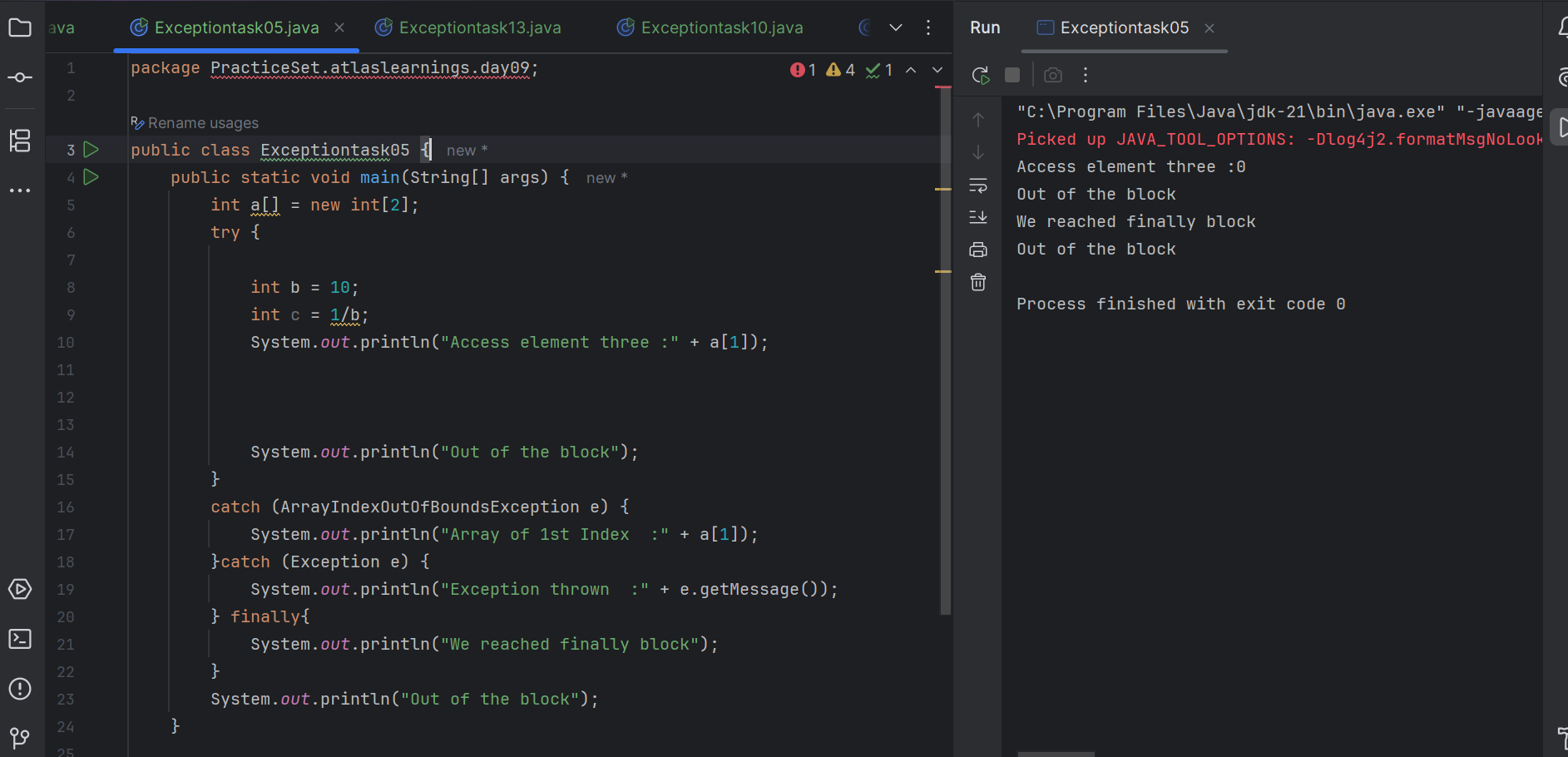
Task -3:



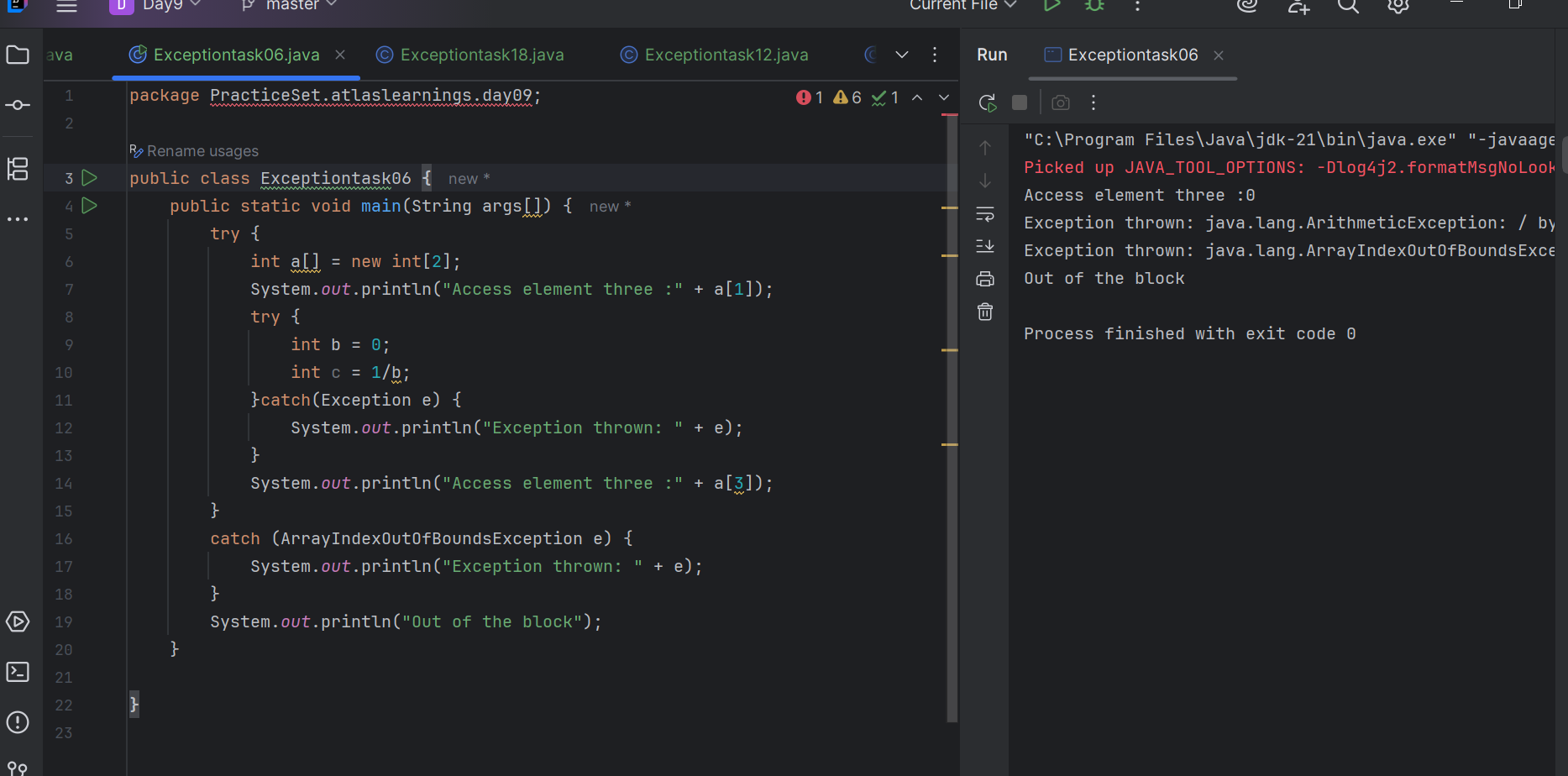
Task -4:



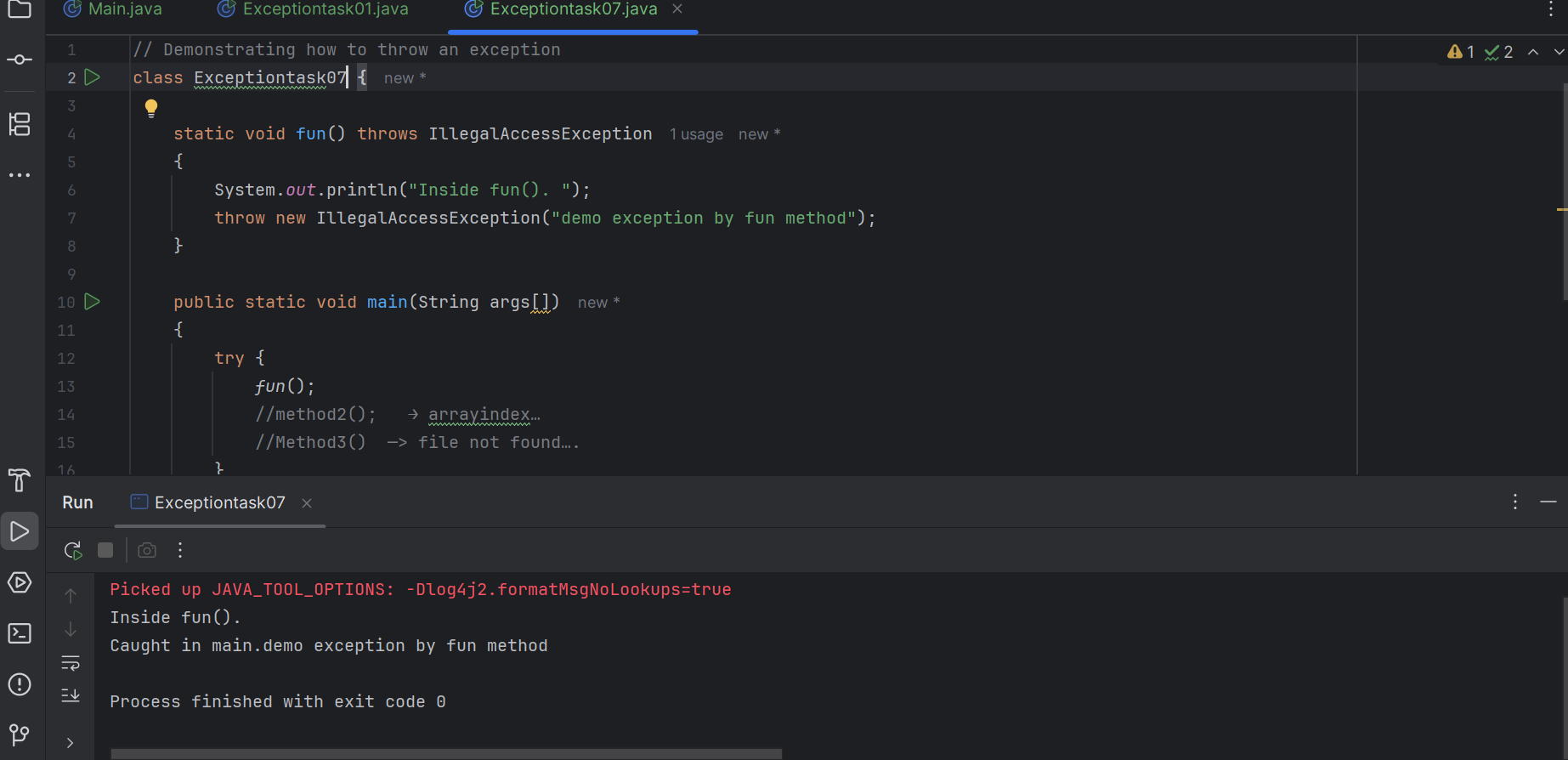
Task -5:



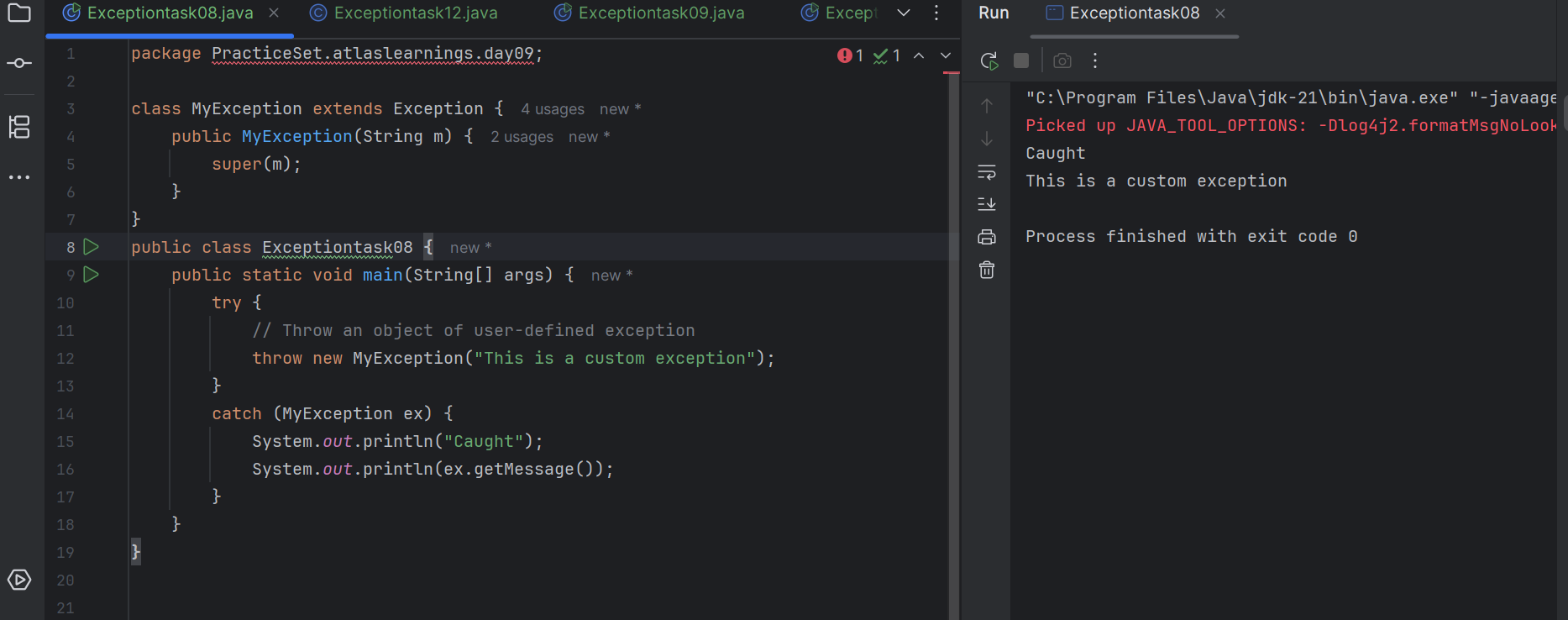
Task -6:



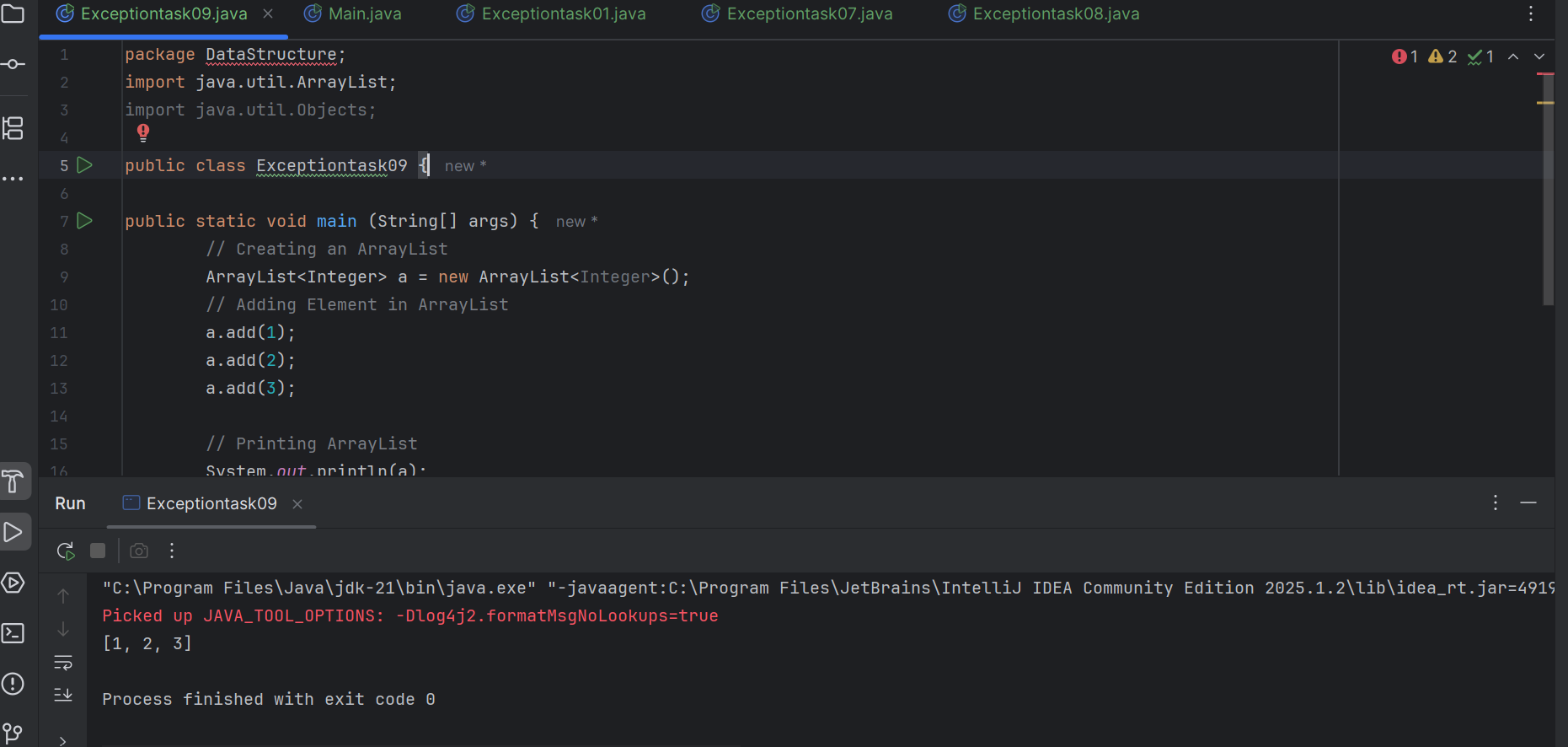
Task - 7:

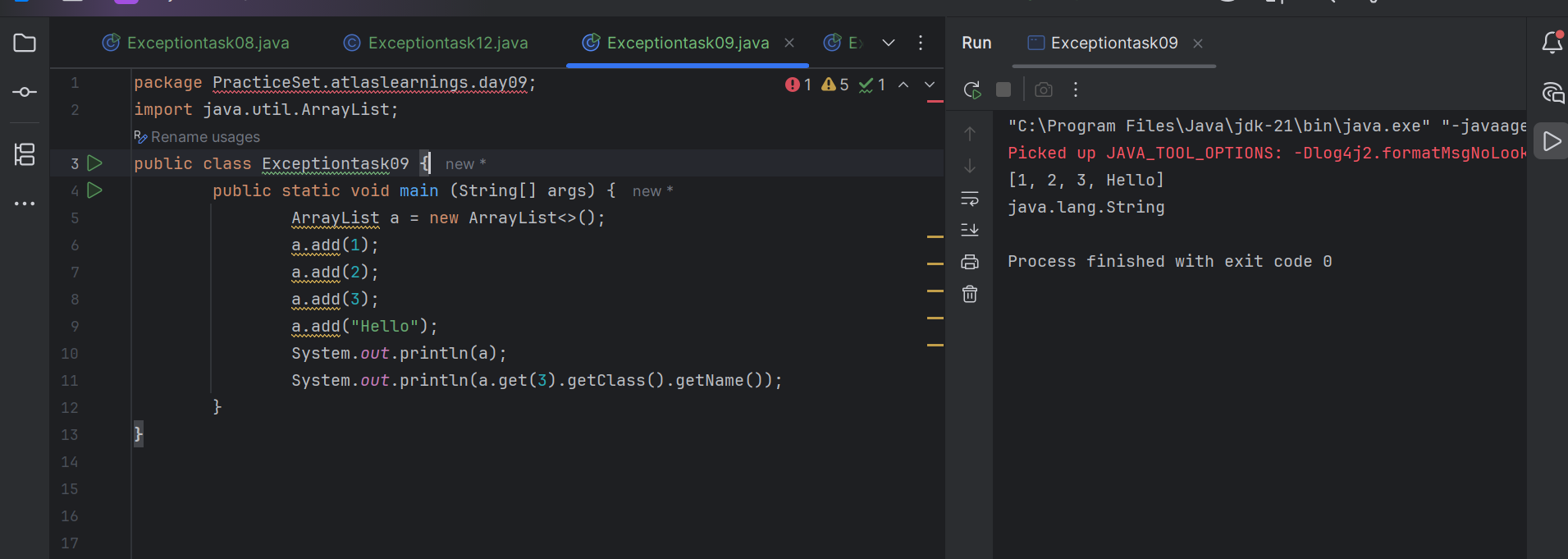


Task -8:

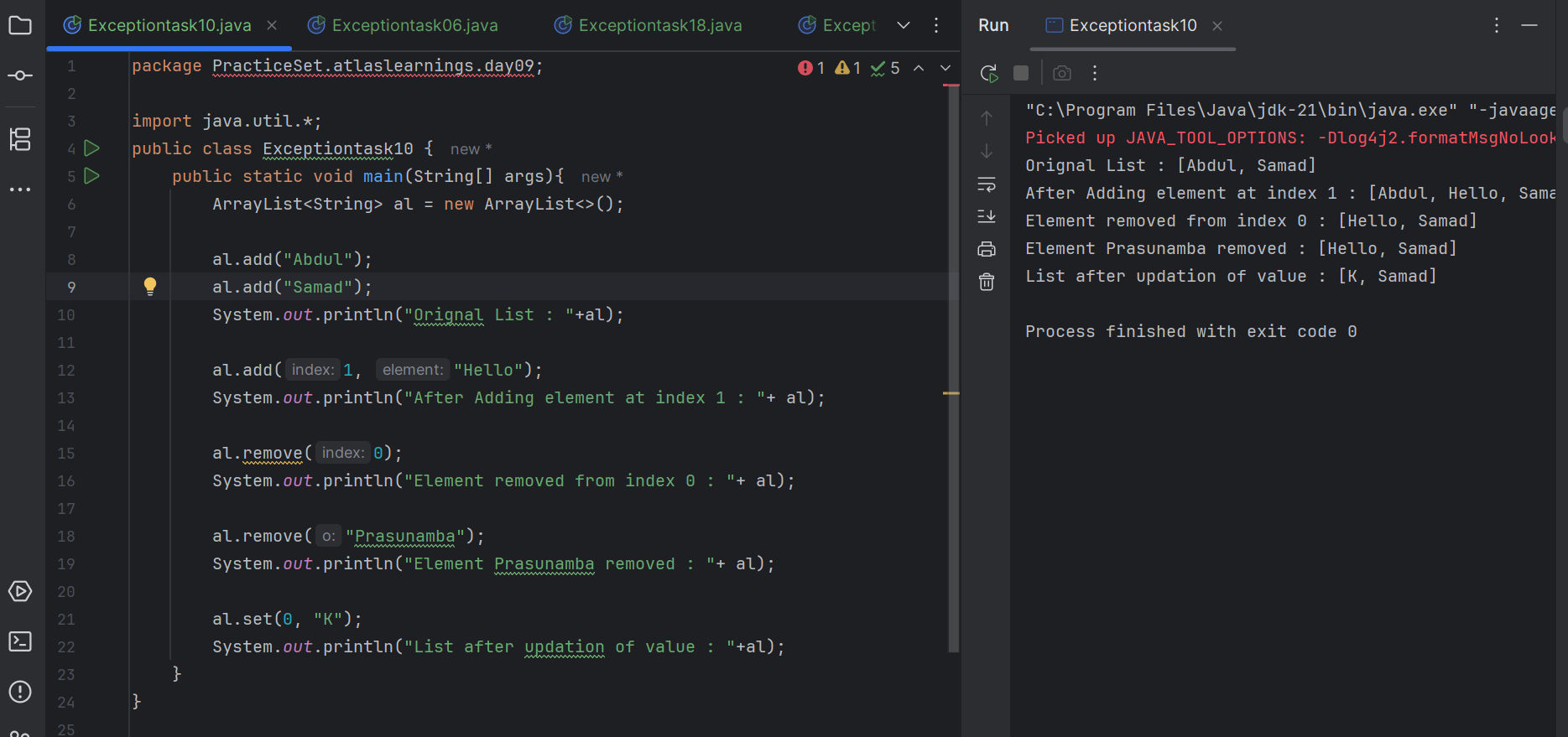


Task -9:

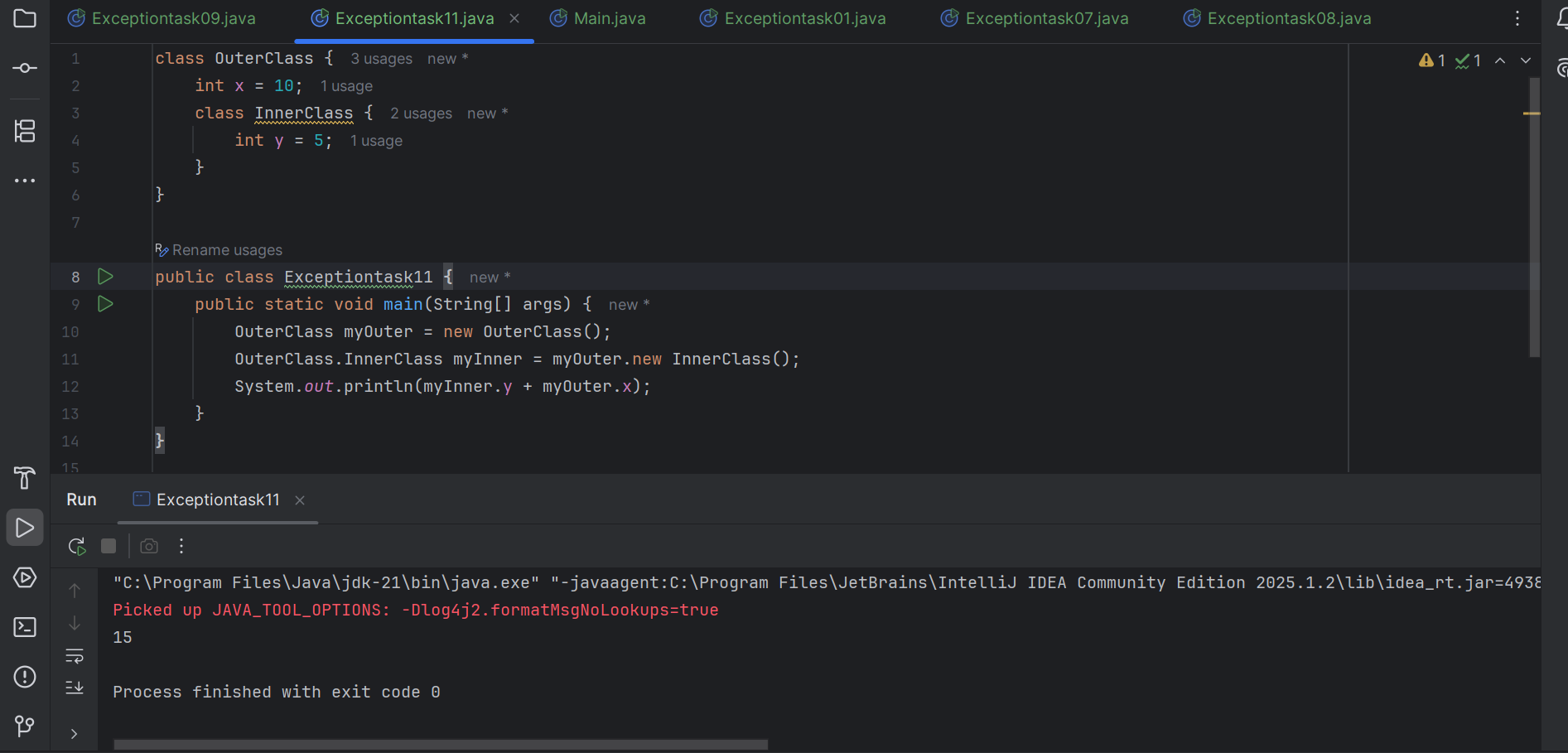




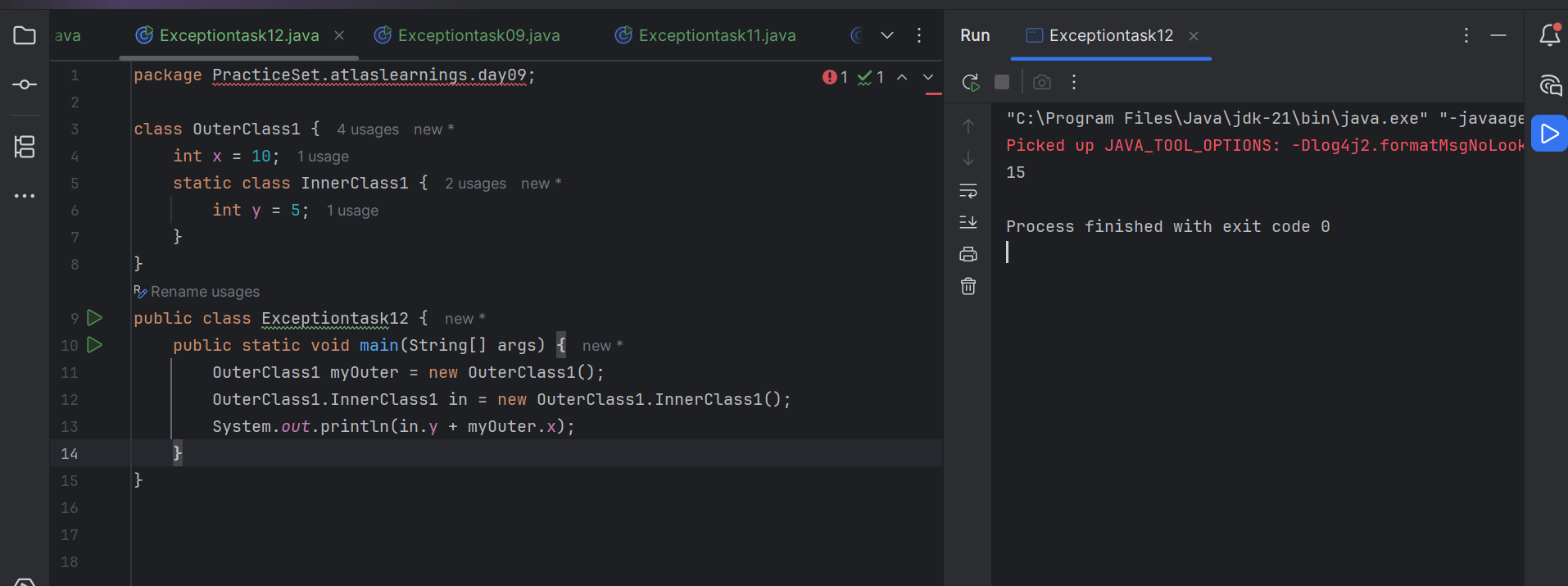
Task -10:



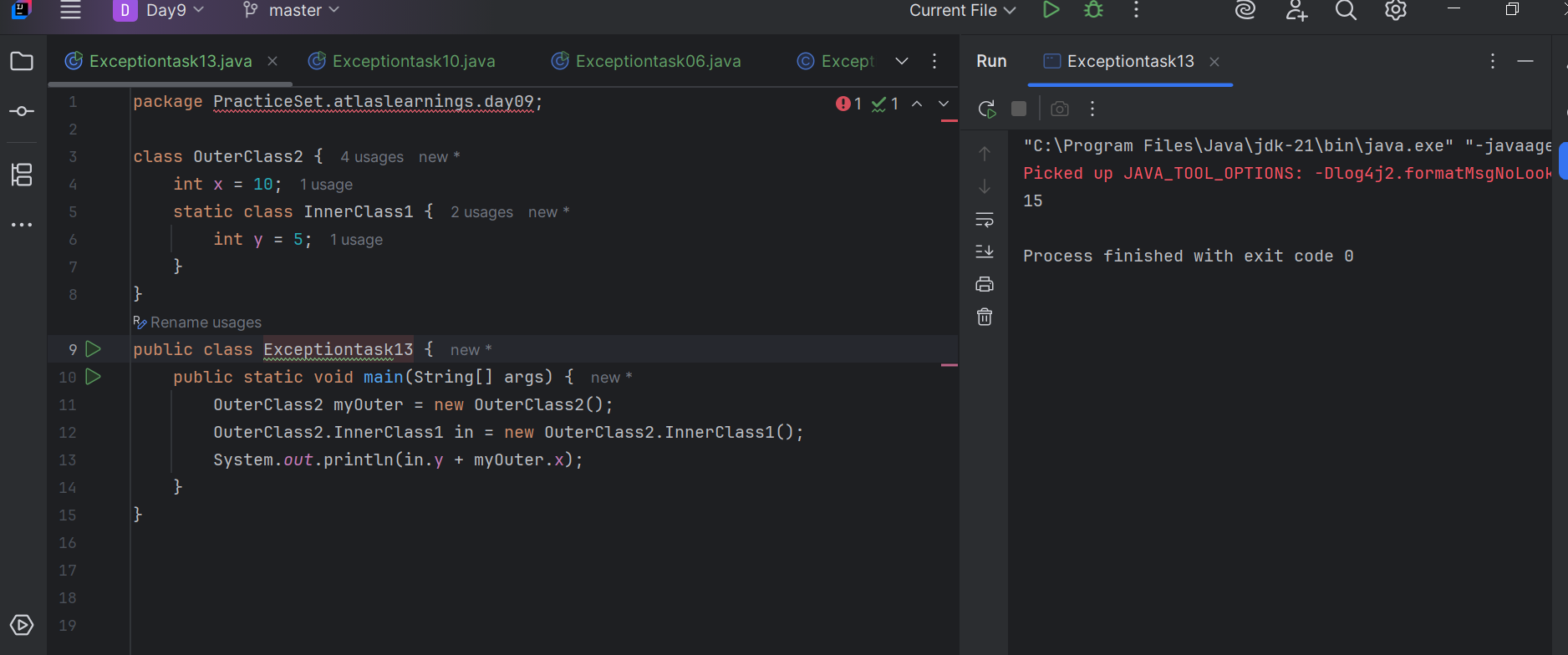
Task -11:



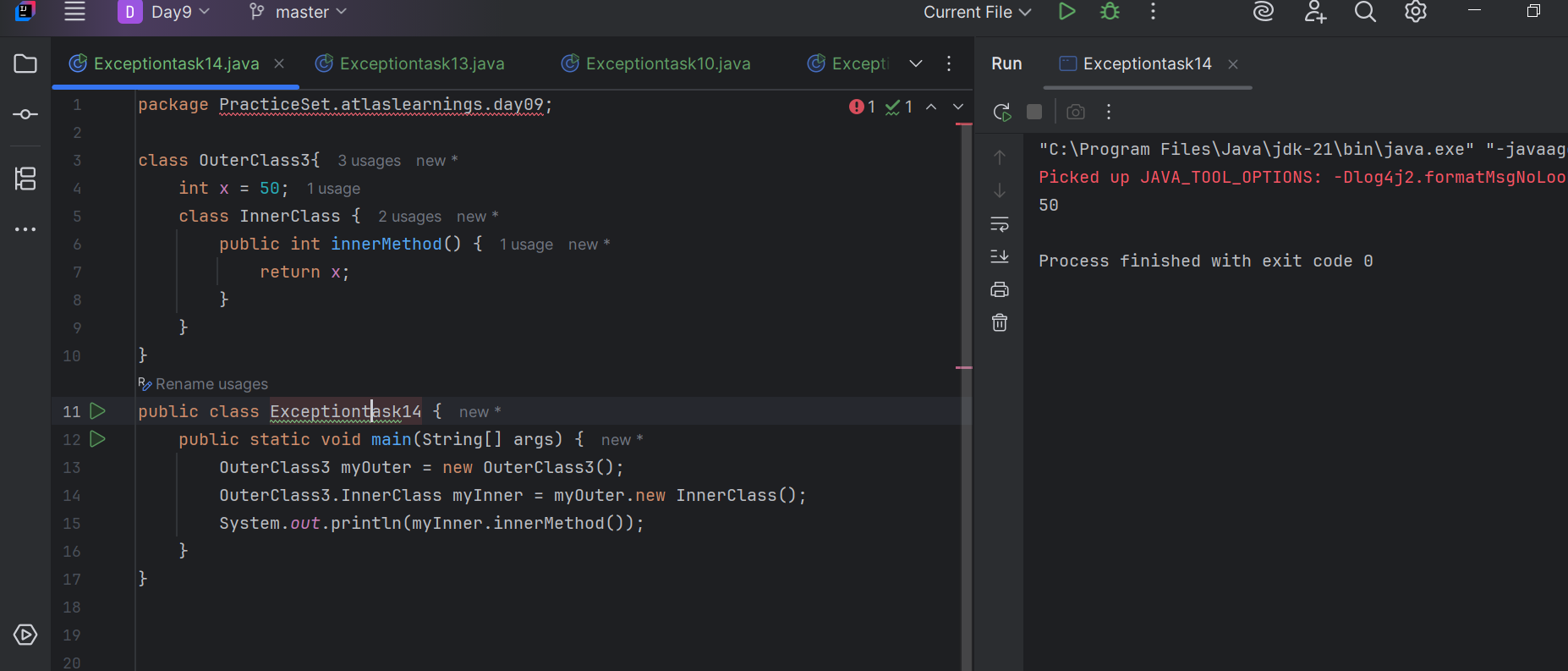
Task -12:



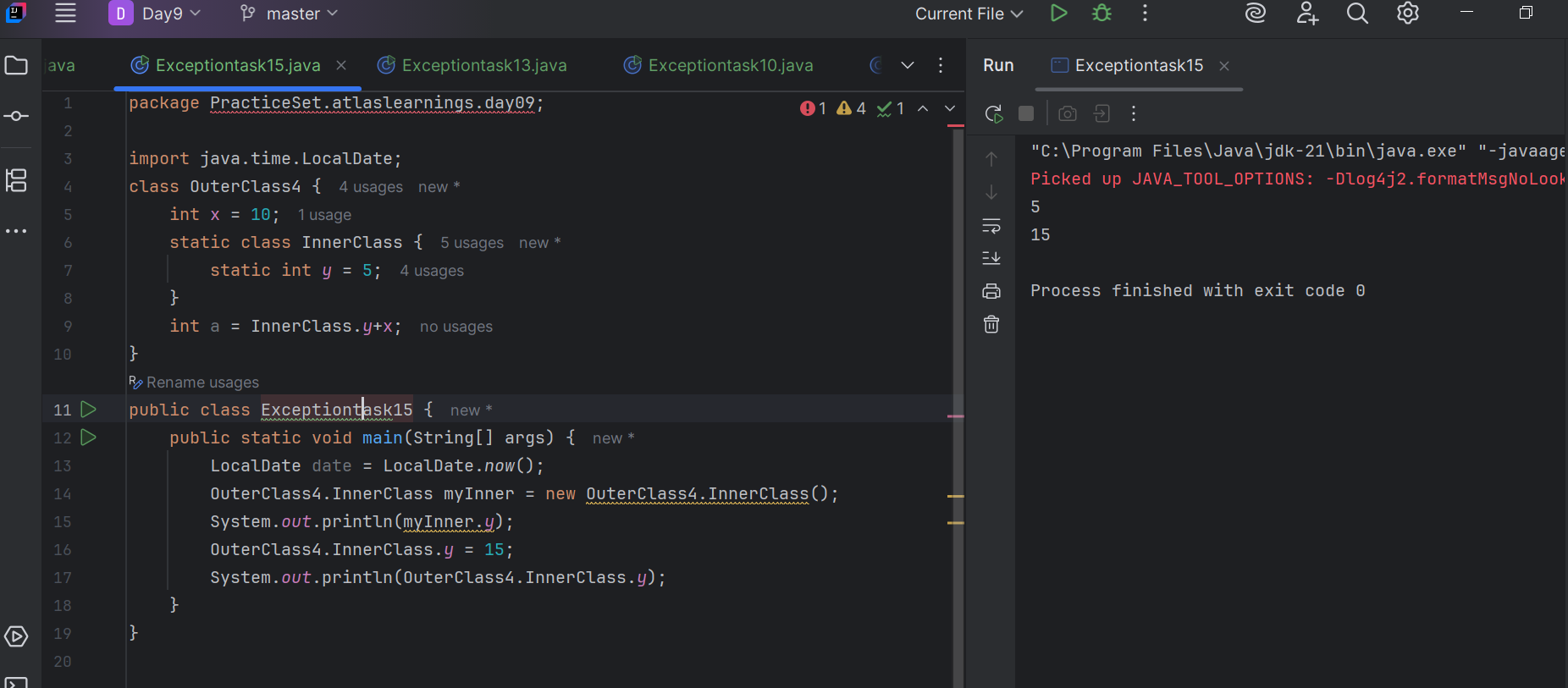
Task -13:



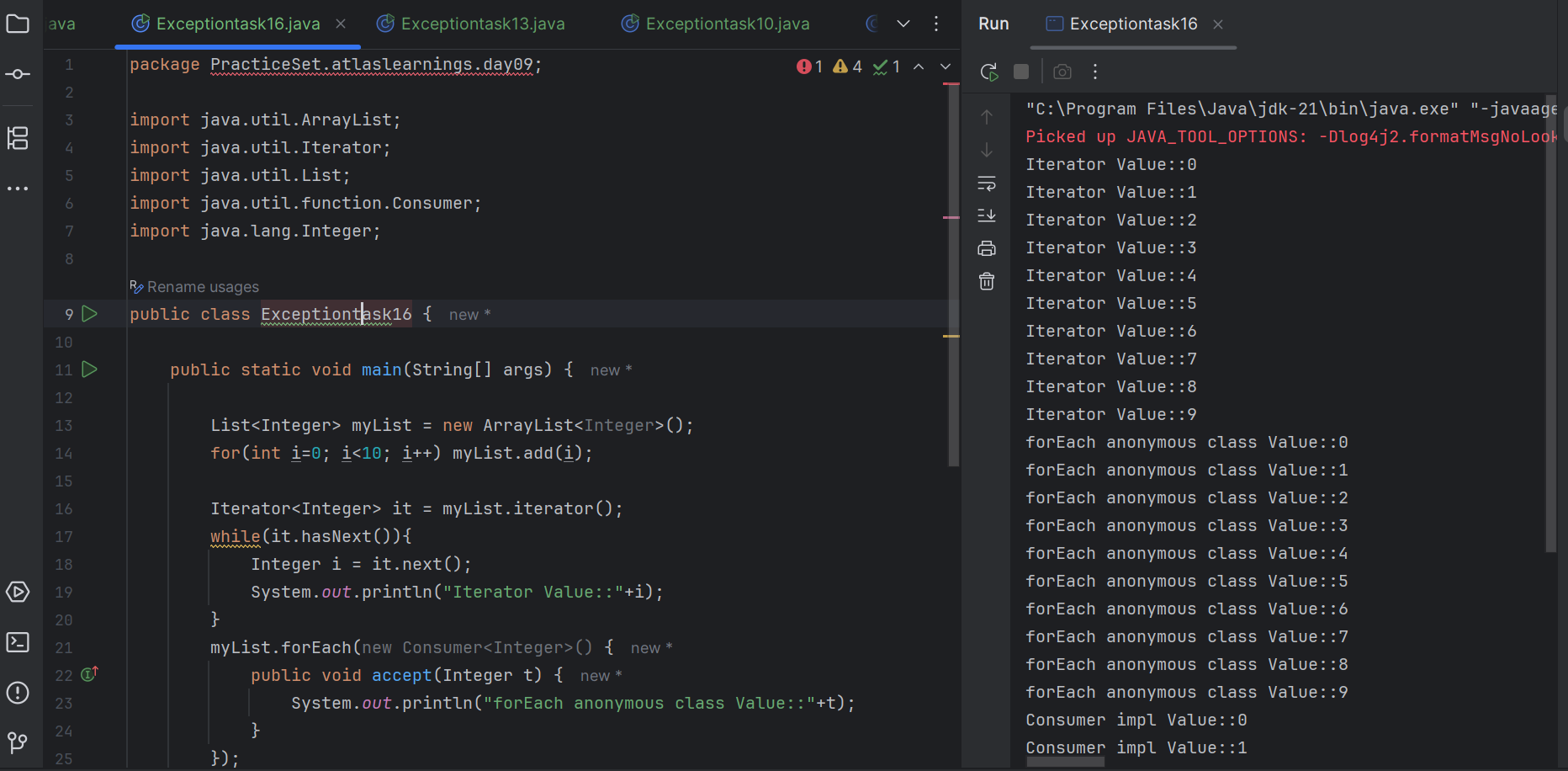
Task- 14:

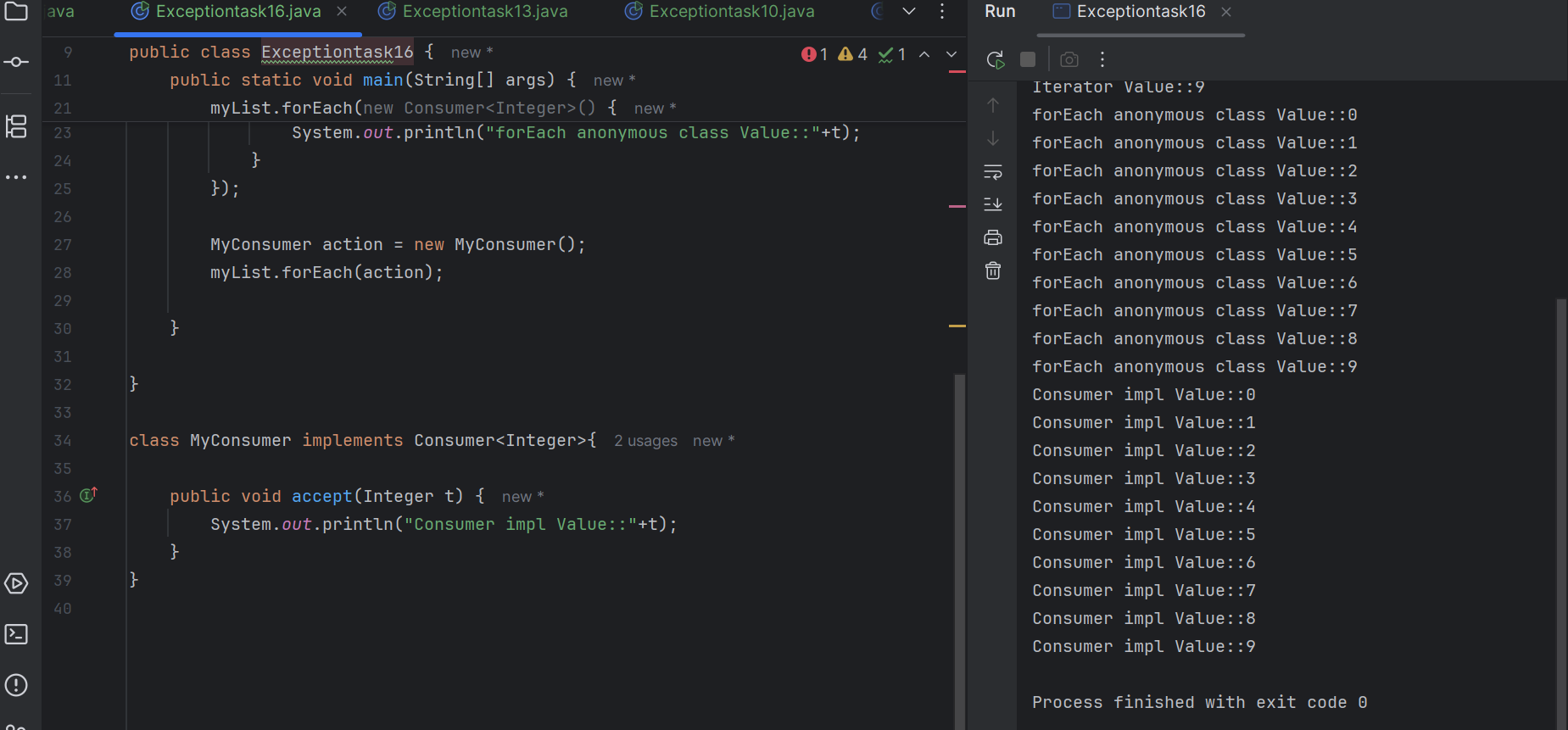


Task -15:



Task -16:





Task-17:

What are the features of Java 8 ?

## 1. Lambda Expressions

Lambda expressions let you write functional-style code — concise, readable, and less boilerplate.

### Before Java 8:

java

Runnable r = new Runnable() {

public void run() {

System.out.println("Running...");

}

};

### Java 8:

java

Runnable r = () -> System.out.println("Running...");

## 2. Functional Interfaces

A functional interface has exactly one abstract method. Example: Runnable, Comparator, Predicate, etc.

java

@FunctionalInterface

interface MyFunction {

void apply();

}

## 3. Streams API

A powerful abstraction for processing collections in a declarative, functional way.

java

List<String> names = Arrays.asList("Alice", "Bob", "Charlie");

names.stream()

.filter(name -> name.startsWith("A"))

.forEach(System.out::println); // prints Alice

## 4. Default Methods in Interfaces

Interfaces can now have default method implementations, improving backward compatibility.

java

interface MyInterface {

default void show() {

System.out.println("Default implementation");

}

}

## 5. Method References

A shortcut to refer to a method by name, using :: instead of writing a full lambda.

java

names.forEach(System.out::println);

## 6. Optional Class

A container object that may or may not contain a non-null value — avoids null pointer exceptions.

java

Optional<String> name = Optional.ofNullable(null);

System.out.println(name.orElse("Default Name"));

## 7. New Date and Time API (java.time)

Replaces the old Date and Calendar classes with a modern, immutable, and fluent API.

java

LocalDate today = LocalDate.now();

LocalDate birthday = LocalDate.of(2000, Month.JUNE, 13);

## 8. Nashorn JavaScript Engine

Java 8 introduced the Nashorn engine for running JavaScript inside the JVM (later removed in Java 15).

## 9. Collectors and Collectors.toList()

Part of the Streams API — used to collect elements into lists, sets, maps, etc.

java

List<String> upper = names.stream()

.map(String::toUpperCase)

.collect(Collectors.toList());

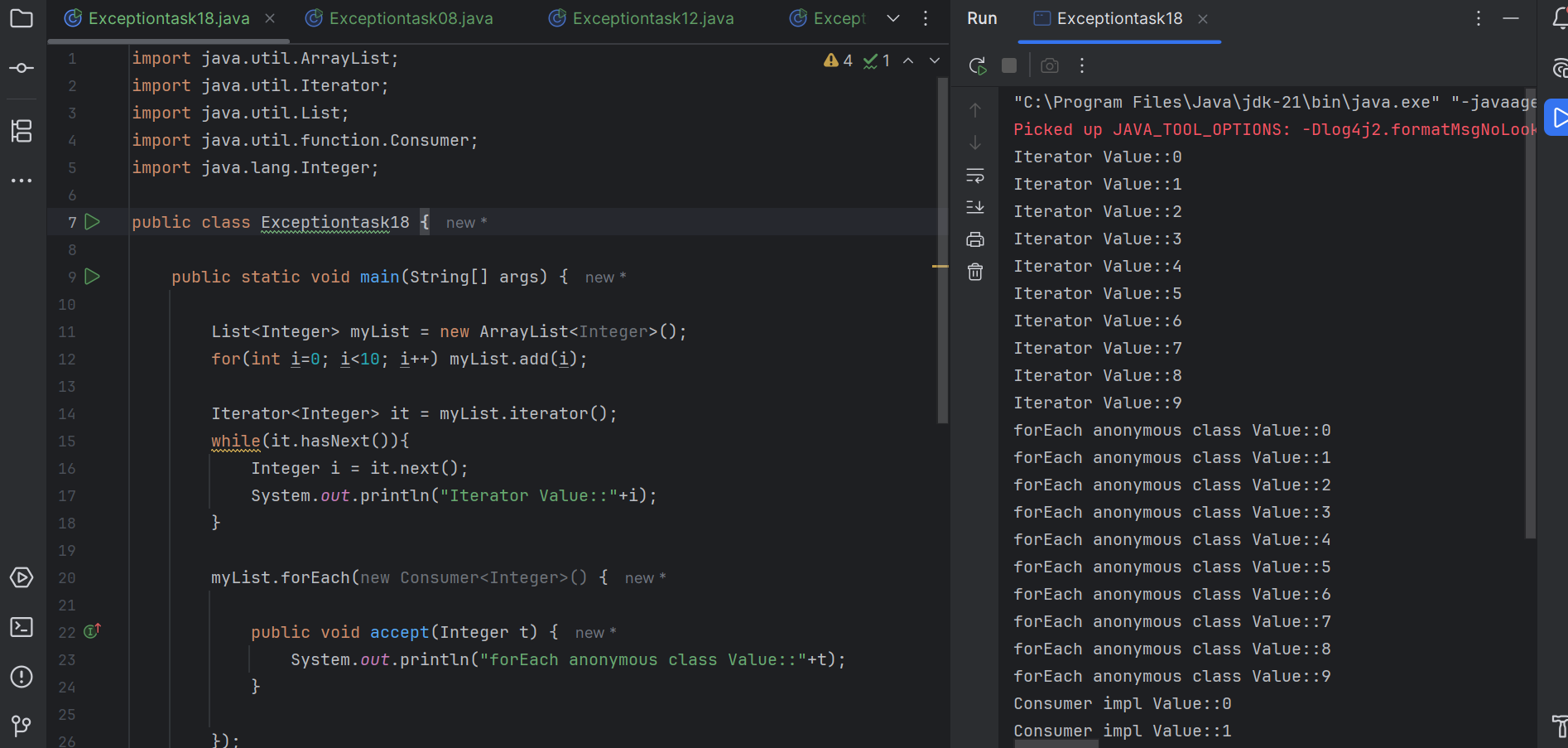
## 10. Parallel Streams:

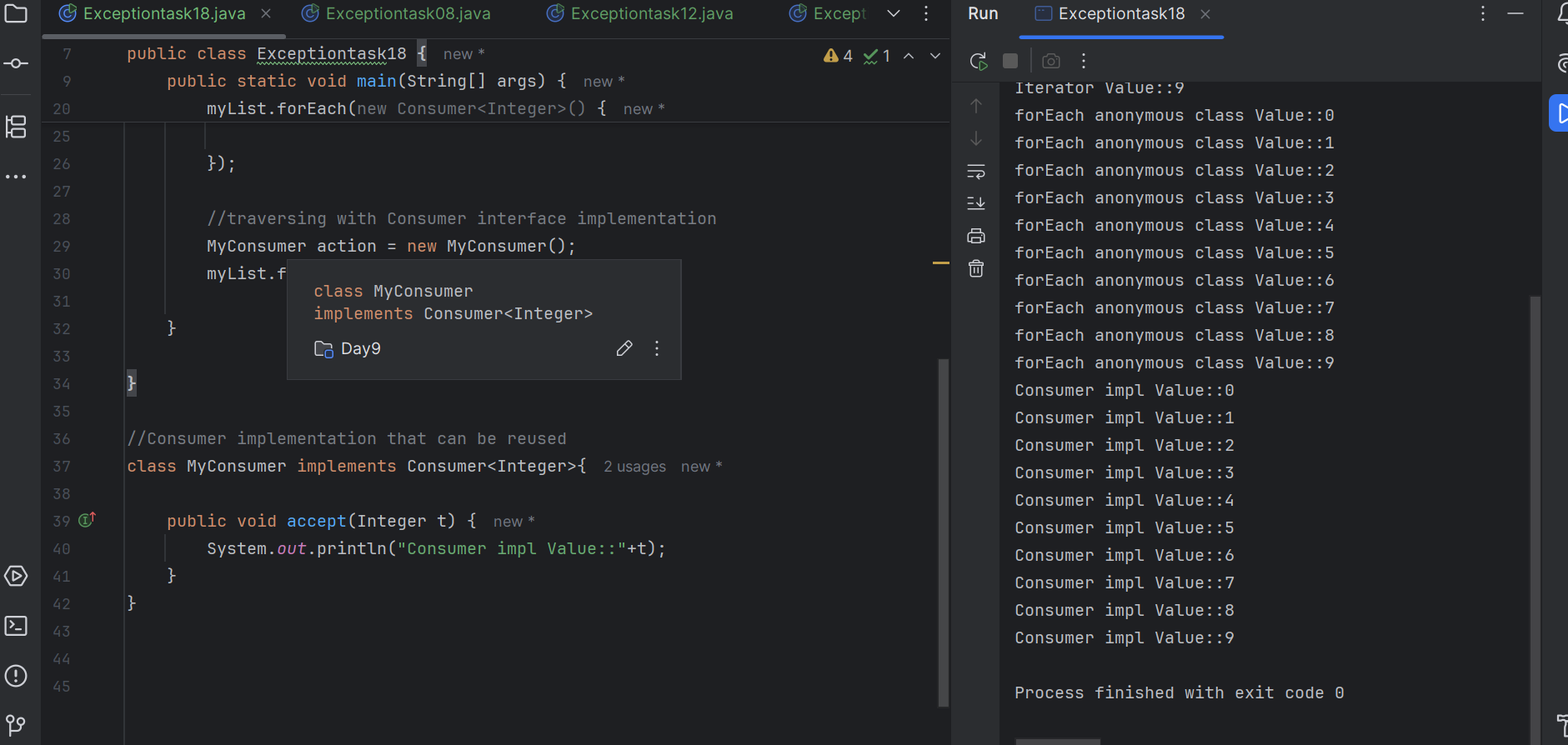
Easily process streams in parallel threads using .parallelStream():

Java

names.parallelStream().forEach(System.out::println);

Task -18:





Task -20:

