CSCI 132:
Basic Data Structures and Algorithms

More Java Constructs, Java Generics, Software Testing

Reese Pearsall

Spring 2025

Announcements

Lab 13 due **tomorrow** @ 11:59 pm

- Course Evaluation

Wednesday will be a help session for Program 5

- No lecture

Final Exam- **Monday May 5th**

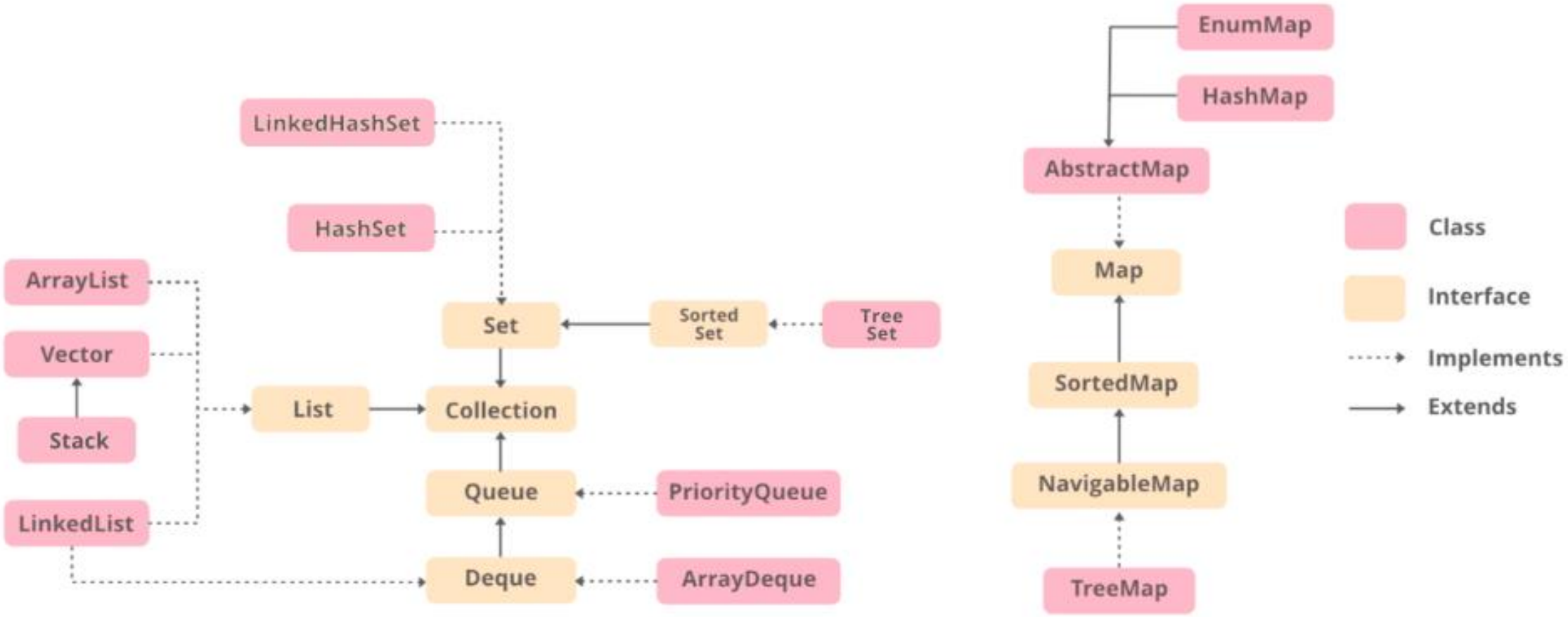
- 2:00 PM – 3:50 PM (Same Classroom)
- Same format as midterm exam

Program 5 due **Sunday May 4th**

- Rubber Duck Extra credit screenshot due by Friday

Please look at the gradebook this week and let someone know if you are missing a grade

Data Structure Class Hierarchy in Java



Instead of writing many `if/else` statements, you can use the `switch` statement

The `switch` statement selects one of many code blocks to be executed

```
int day = 4;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
       System.out.println("Wednesday");
       break;
    default:
        System.out.println(“???");
```

These can be efficient when working with many possible conditions. They serve the same purpose as if statements, but are *slightly* more efficient

Wrapper Classes

Every primitive data type in Java has a **Wrapper Class**

A Wrapper Class is a way to represent a primitive data type as a Java Object

Primitive Type	**Wrapper Class**
`int`	`Integer`
`double`	`Double`
`char`	`Character`

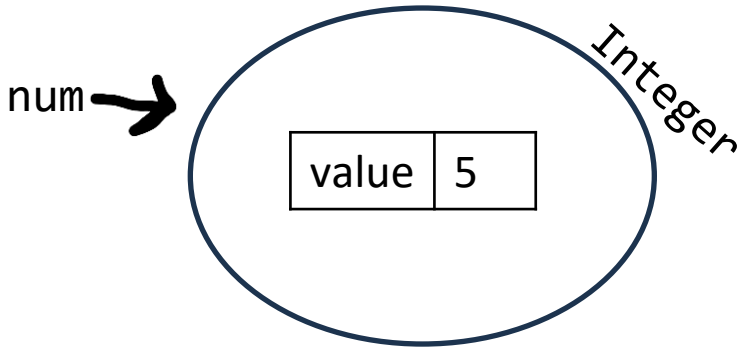
Wrapper classes also provide lots of helpful built-in methods

```
Integer.compareTo(…)
Integer.ParseInt(…)
Integer.toString(..)
```

```
int num = 5;
```

num	5

```
Integer num = 5;
```




```
int num = null;
```

```
Integer num = null;
```

Let's go back to when we were writing our own Linked List and Node class

For example, this Linked List could <u>only</u> hold Strings

```
public class Node {

    private String name;
    private Node next;

    public Node(String c) {
        this.name = c;
        this.next = null
    }
   …
}
```

Let's go back to when we were writing our own Linked List and Node class

For example, this Linked List could <u>only</u> hold Strings

```
public class Node {

    private String name;
    private Node next;

    public Node(String c) {
        this.name = c;
        this.next = null
    }
   …
}
```

If we wanted to have Linked List hold Doubles, we would need to modify parts of the Node and LinkedList class

```
public class Node {

    private double value;
    private Node next;

    public Node(double c) {
        this.value = c;
        this.next = null
    }
   …
}
```

Let's go back to when we were writing our own Linked List and Node class

For example, this Linked List could <u>only</u> hold Strings

```
public class Node {

    private String name;
    private Node next;

    public Node(String c) {
        this.name = c;
        this.next = null
    }
   …
}
```

If we wanted to have Linked List hold Doubles, we would need to modify parts of the Node and LinkedList class

```
public class Node {

    private double value;
    private Node next;

    public Node(double c) {
        this.value = c;
        this.next = null
    }
   …
}
```

It would be nice if we could allow our Linked List to hold **any type of data** without needing to modify the source code of our classes

Let's go back to when we were writing our own Linked List and Node class

For example, this Linked List could <u>only</u> hold Strings

```
public class Node {

    private String name;
    private Node next;

    public Node(String c) {
        this.name = c;
        this.next = null
    }
   …
}
```

If we wanted to have Linked List hold Doubles, we would need to modify parts of the Node and LinkedList class

```
public class Node {

    private double value;
    private Node next;

    public Node(String c) {
        this.name = c;
        this.next = null
    }
   …
}
```

It would be nice if we could allow our Linked List to hold **any type of data** without needing to modify the source code of our classes→ We can achieve this using **Java generics**

We can **embed** a class within another class (although I don't recommend doing this unless the class is very small and/or the classes are strongly related to each other)

```
public class GenericLinkedList {

    public class Node<E>{

        E data;

        Node<E> next;

        public Node(E data){
            this.data = data;
            this.next = null;
        }

        public E getData() {
            return this.data;
        }

        public Node getNext() {
            return this.next;
        }

    }


    private Node head;
    private int size;

    public GenericLinkedList() {
        this.head = null;
        this.size = 0;
    }

    public <E> void add(E newData) {
```

The data can be *any object*

When we create a Node object, we will give it some data type

getData() will now return some generic object E

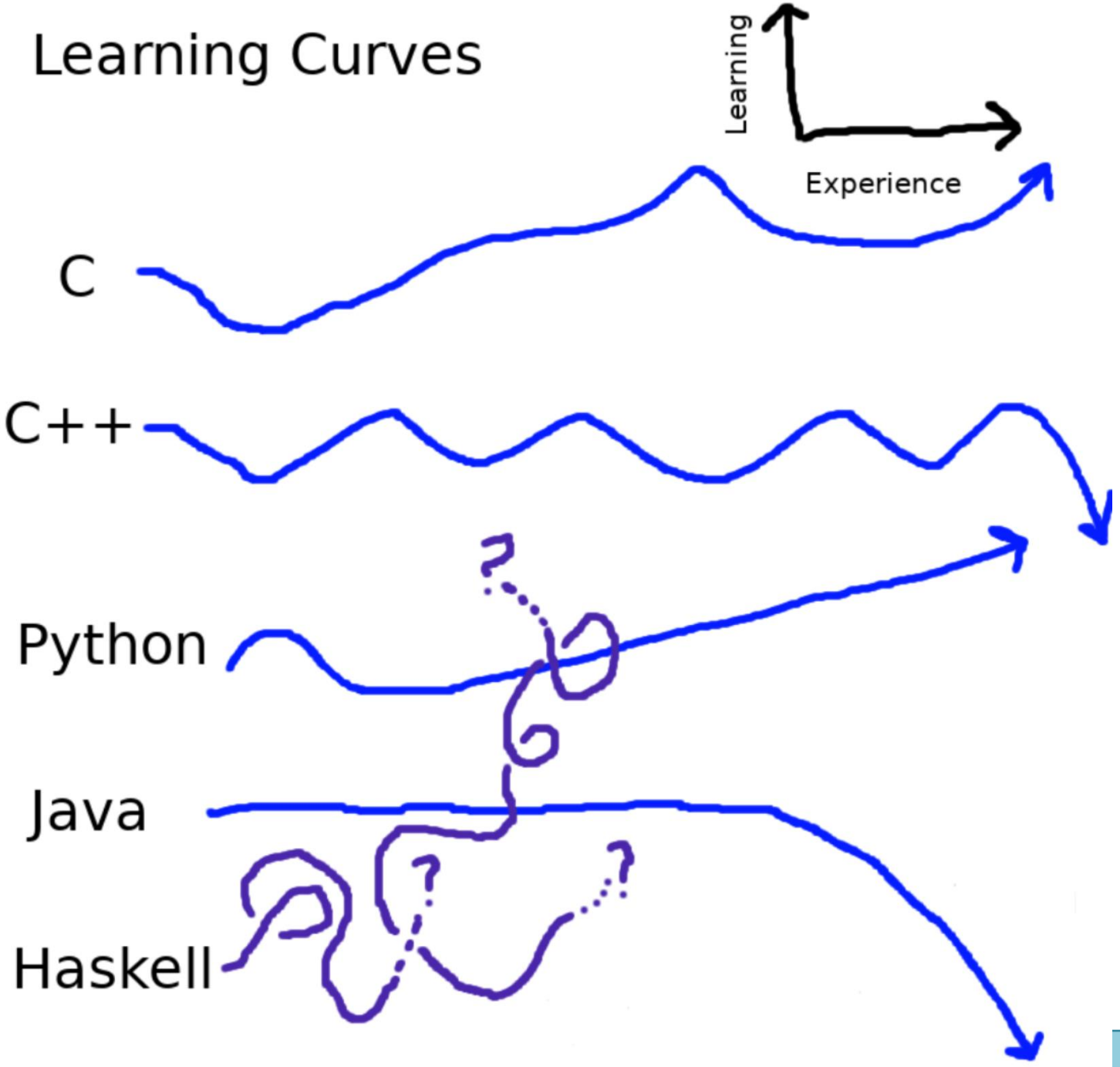
Start of Linked List class

<E> is used to indicate that this Node class will hold a **Generic object**. It can be *any* object

This is very helpful for cases when we might not know what data type we will be working with

<T> is also a value used to indicate a generic object

Learning Curves



```java
this.graph = Files.lines(file_path) Stream<String>
        .map( String line -> {
            Matcher matcher = vertexPattern.matcher(line); // For each line, find all vertex matches
            List<String> vertices = new ArrayList<>();
            while (matcher.find()) {
                vertices.add(matcher.group()); // Add the matched vertex string "<...>"
            }
            return vertices; // Return the list of vertices found on this line
        }) Stream<List<...>>
        .filter( List<String> vertices -> vertices.size() == 2) // IMPORTANT: Only process lines that yielded exactly two vertices | TODO: add hand
        .collect(Collectors.groupingBy(
                List<String> vertices -> vertices.get(0),    // Key: The first vertex (source)
                HashMap::new,                   // Use HashMap for the main map structure
                Collectors.mapping(             // Map the downstream elements before collecting
                        List<String> vertices -> vertices.get(1), // Get the second vertex (target)
                        Collectors.toList()          // Collect the target vertices into a List (defaults to ArrayList)
                )
                //TODO: This should also count the occurrences of repeat edges to account for parallel edges
        ));
this.graph.keySet().forEach( String vertex -> {System.out.println(vertex + " -> " + this.graph.get(vertex));});
```

Software testing involves verifying and validating that a software application is free of bugs/errors and that it meets the technical requirements.

Today we will be covering **unit testing**

Unit testing is preformed by developers during coding

This method tests individual components of an application to identify bugs early

We will be using JUnit, a testing framework built into eclipse

JUnit

Test Driven Development (TDD)

Might see it on job listings

Very popular software development approach where you write tests for a feature before writing the actual code that implements the feature

Main idea is to write automated tests first, then write the code to pass those tests to produce code with fewer bugs

Unit Testing

Unit testing focuses on testing individual components or "units" of code (usually functions) independently from the rest of the program

Focused tests that check if a function returns the correct output given a specific input

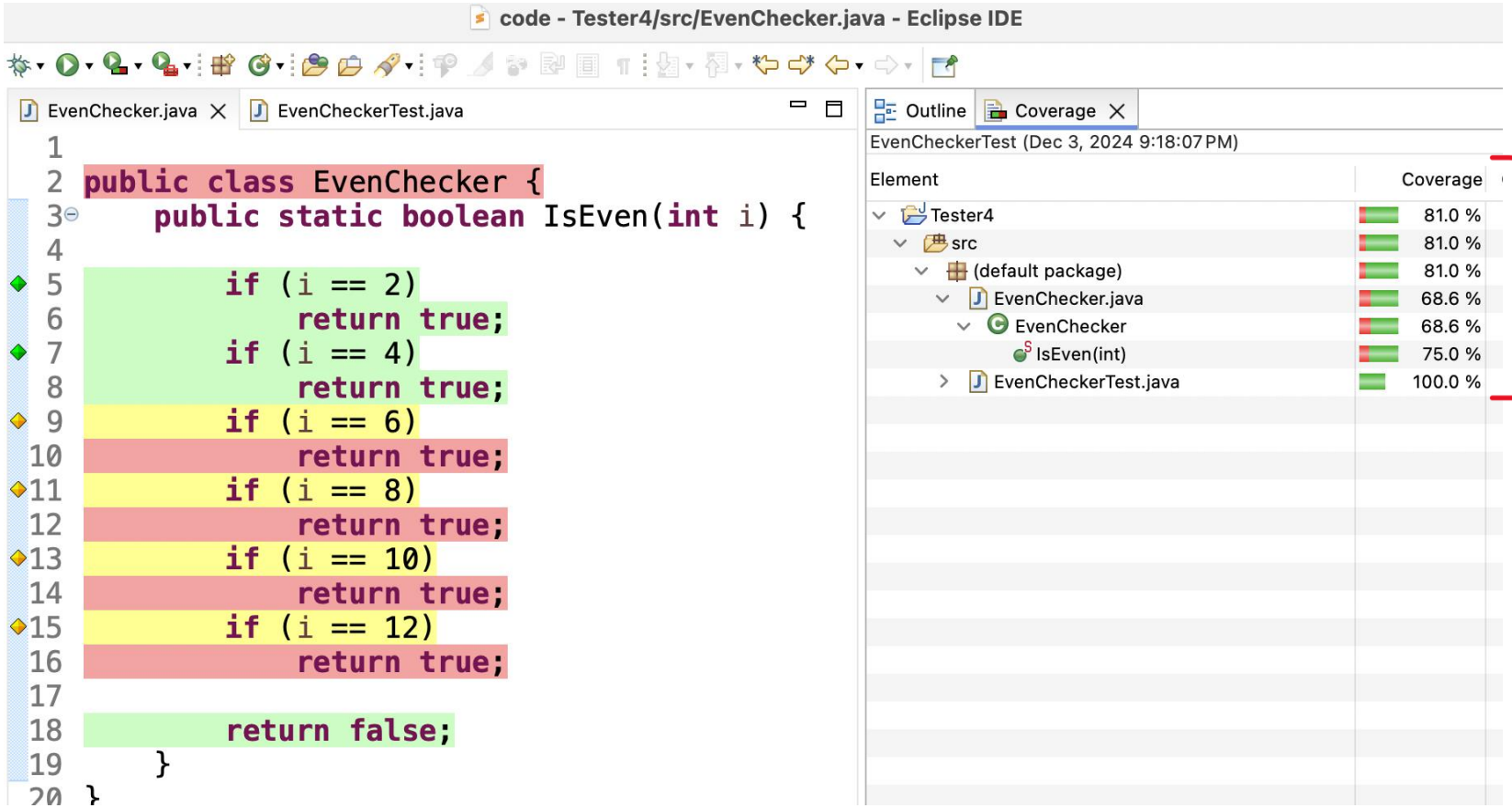
Allows you to check for strange behavior or confirm your program does what it should when it encounters edge cases

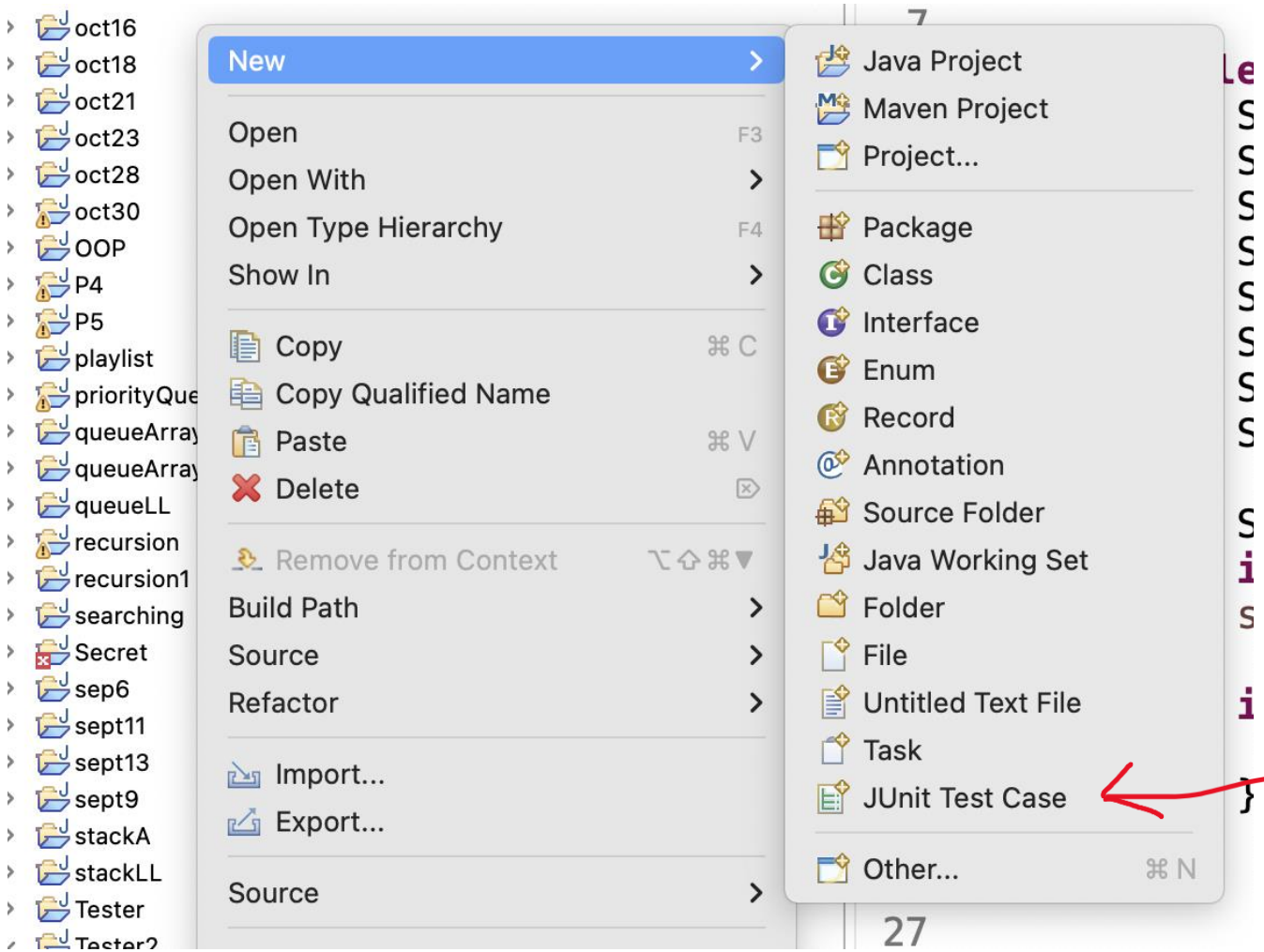
Code Coverage

Code coverage is a metric used in software testing to measure the percentage of your code that is executed when you run your test suite

A percentage value reflects how much of your code is tested within your unit tests

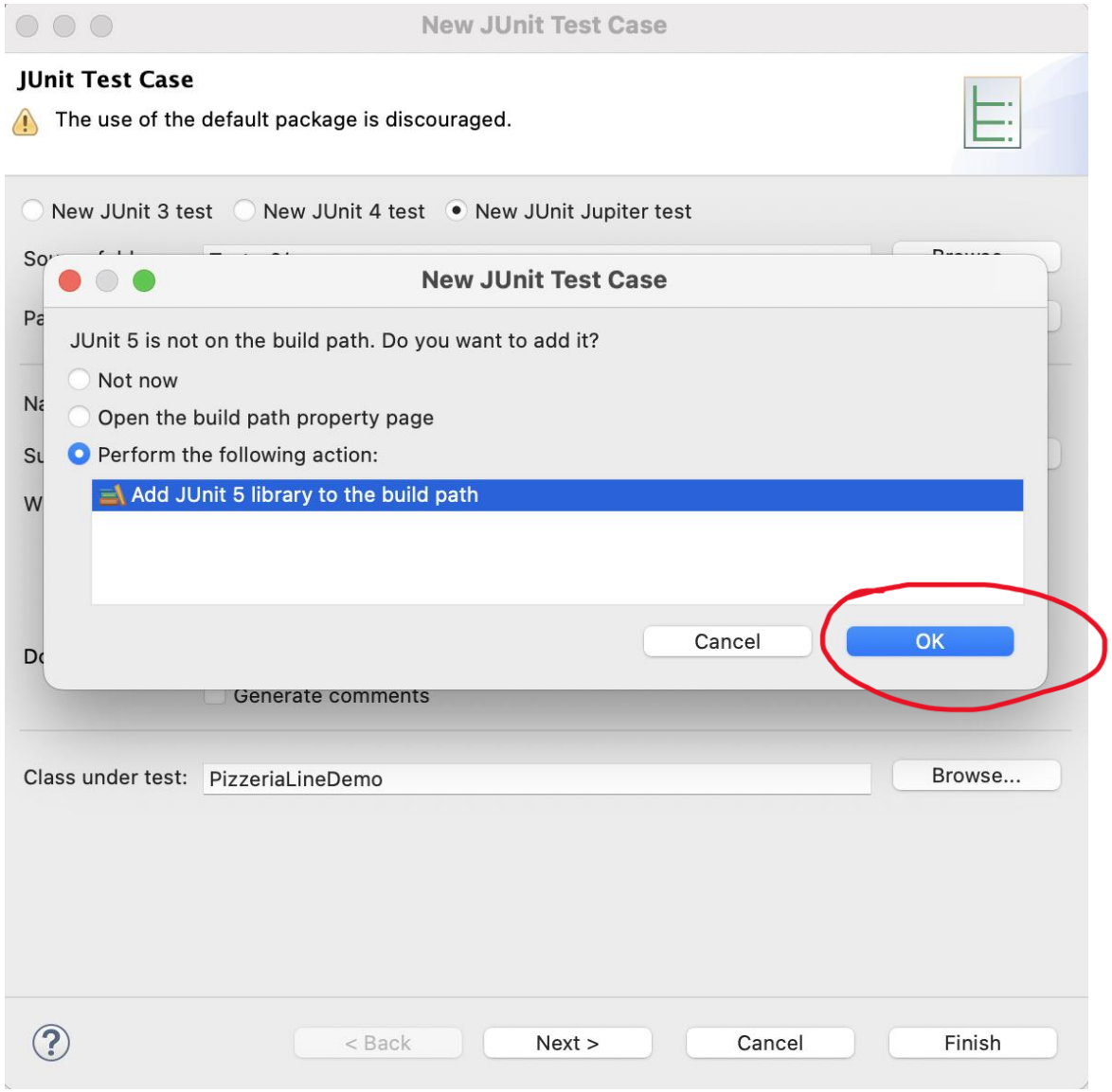
Green → tested
Yellow → partially tested
Red → not tested





Add new JUnit test case

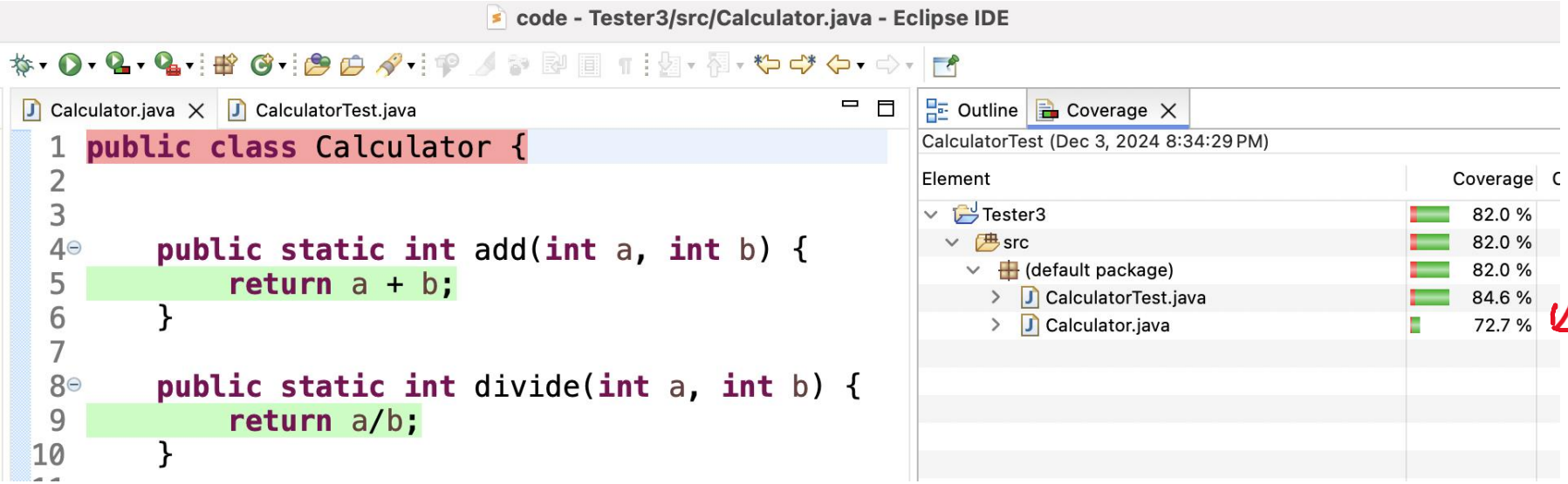
Add the JUnit library to the project path



Code Coverage Requirements

You may encounter situations where you're required to meet a minimum level of test coverage (ex. 70%) before you can release your code

Reaching a certain level of coverage is often a key part of maintaining code quality and meeting standards when working in industry



Calculator.java has 72.7% test coverage

(since we never instantiated a calculator object)