

CSCI 132:

Basic Data Structures and Algorithms

More Java Constructs, Java Generics

Reese Pearsall + Iliana Castillon
Fall 2024

Announcements

Lab 13 due Thursday @ 11:59 pm

- Course Evaluation

Final Exam- **Monday December 9th**

- 2:00 PM – 3:50 PM (Same Classroom)
- Study Guide Posted

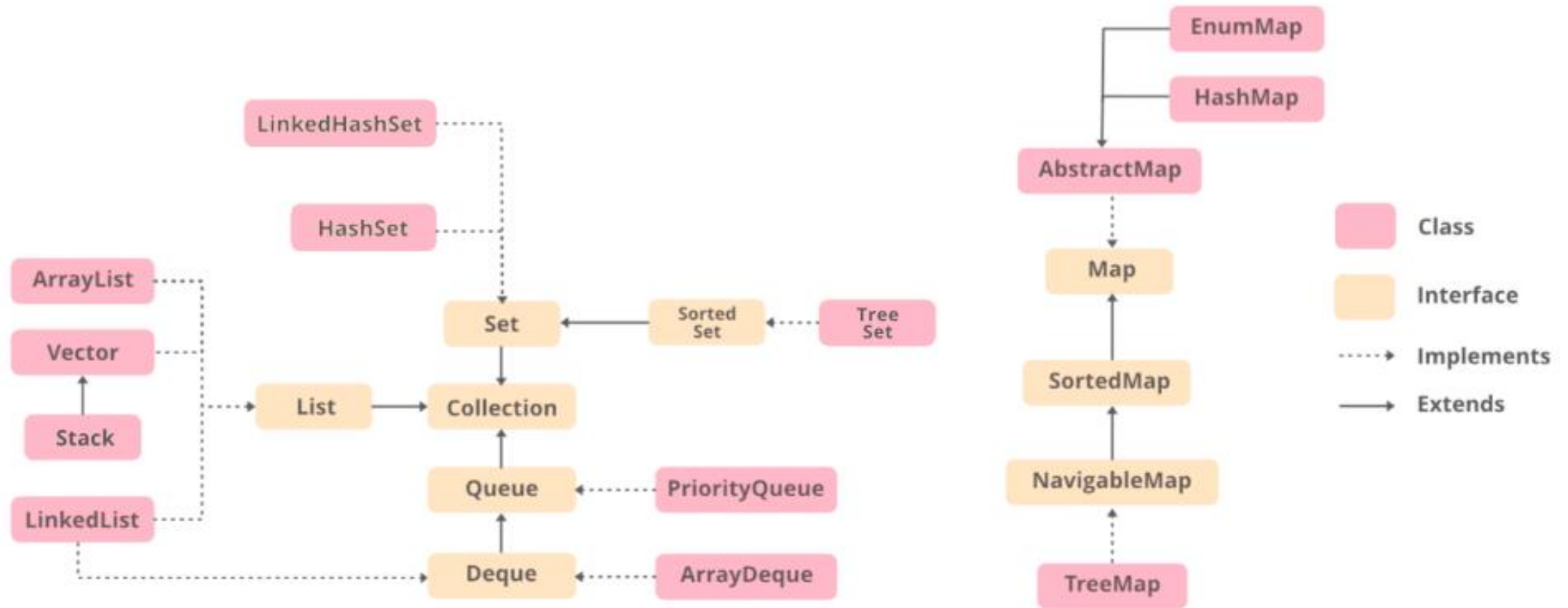
Program 5 due **Sunday December 8th**

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

- Rubber Duck Extra credit screenshot due by Friday



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

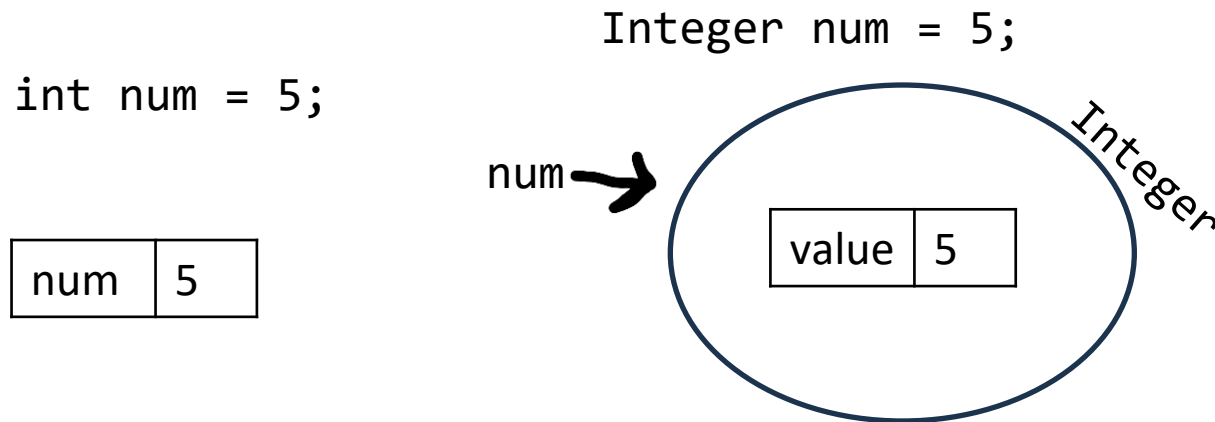
Every primitive data type in Java has a **Wrapper Class**

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 = null; X  
Integer num = null; ✓
```

Java has a built-in `sort` method for Arrays

What sorting algorithm does it use?

<https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>

Java has a built-in `sort` method for Arrays

What sorting algorithm does it use?

Method Detail

`sort`

```
public static void sort(int[] a)
```

Sorts the specified array into ascending numerical order.

Implementation note: The sorting algorithm is a **Dual-Pivot Quicksort** by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers $O(n \log(n))$ performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Parameters:

`a` - the array to be sorted

<https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>

Java has a built-in `sort` method for Arrays

What sorting algorithm does it use?

Method Detail

sort

```
public static void sort(int[] a)
```

Sorts the specified array into ascending numerical order.

Implementation note: The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers $O(n \log(n))$ performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Parameters:

a - the array to be sorted

Python's `.sort()` function uses a hybrid of merge sort and insertion sort, called **Timsort**

Timsort

11 languages

Article Talk

Read Edit View history Tools

From Wikipedia, the free encyclopedia

Timsort is a [hybrid, stable sorting algorithm](#), derived from [merge sort](#) and [insertion sort](#), designed to perform well on many kinds of real-world data. It was implemented by [Tim Peters](#) in 2002 for use in the [Python programming language](#). The algorithm finds subsequences of the data that are already ordered (runs) and uses them to sort the remainder more efficiently. This is done by merging runs until certain criteria are fulfilled. Timsort has been Python's standard sorting algorithm since version 2.3. It is also used to sort arrays of non-primitive type in [Java SE 7](#),^[4] on the [Android platform](#),^[5] in [GNU Octave](#),^[6] on [V8](#),^[7] [Swift](#),^[8] and [Rust](#).^[9]

It uses techniques from Peter McIlroy's 1993 paper "Optimistic Sorting and Information Theoretic Complexity".^[10]

Timsort

Class	Sorting algorithm
Data structure	Array
Worst-case performance	$O(n \log n)$ ^{[1][2]}
Best-case performance	$O(n)$ ^[3]
Average performance	$O(n \log n)$
Worst-case space complexity	$O(n)$

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

For example, this Linked List could only 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 only 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 only 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 only 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;
```

← The data can be
any object

```
        Node<E> next;
```

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

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

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

getData() will now return some generic object E

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

```
    }
```

```
    private Node head;
    private int size;
```

Start of Linked List class

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

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

<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

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

The logo for JUnit, featuring the letters 'J' and 'U' in a large, green, serif font, and the letters 'nit' in a smaller, red, serif font.

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

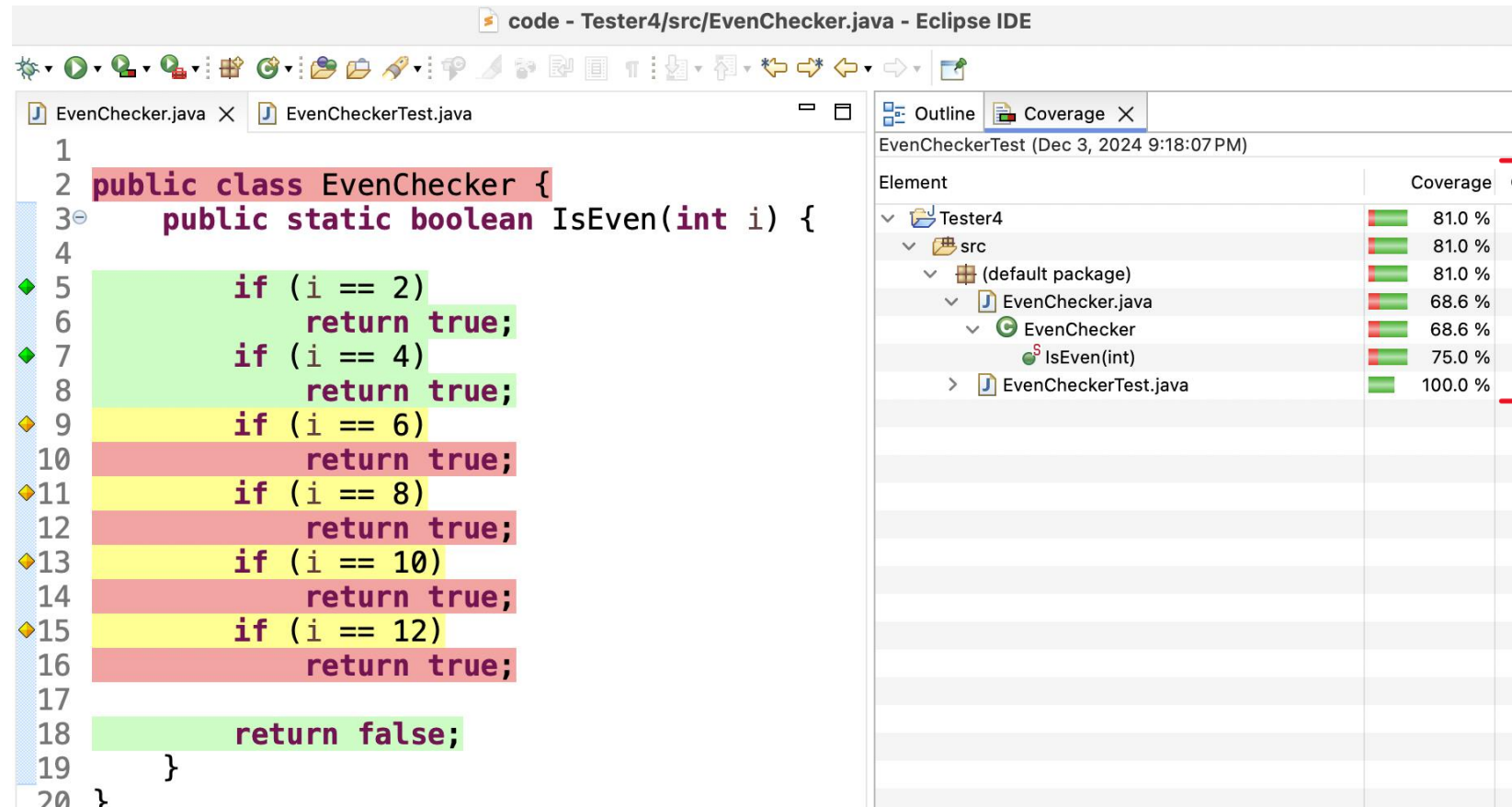
Allows you to check for strange behavior or confirm your program does what it should when it encounters edge cases

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

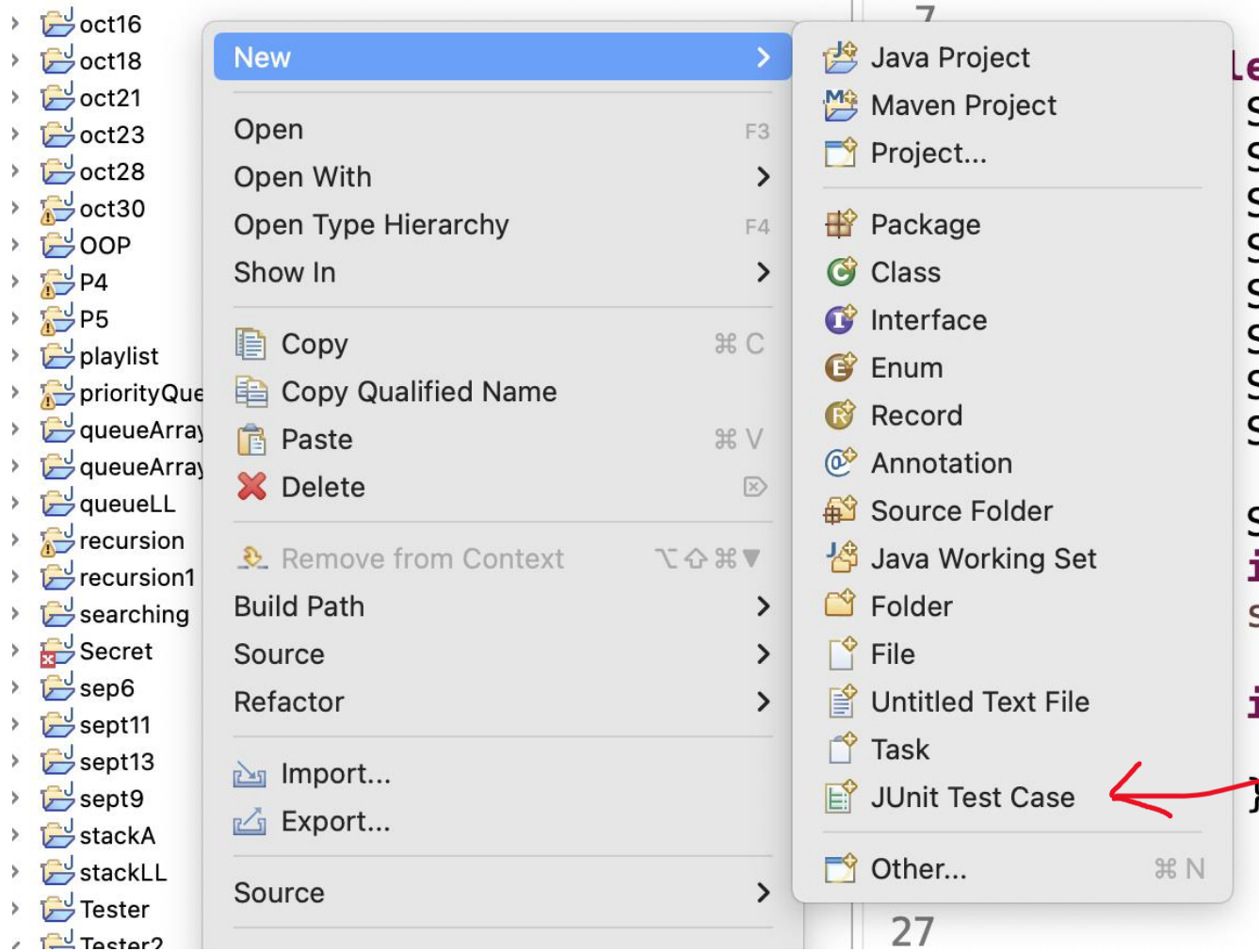


code - Tester4/src/EvenChecker.java - Eclipse IDE

```
1
2 public class EvenChecker {
3     public static boolean IsEven(int i) {
4
5         if (i == 2)
6             return true;
7         if (i == 4)
8             return true;
9         if (i == 6)
10            return true;
11        if (i == 8)
12            return true;
13        if (i == 10)
14            return true;
15        if (i == 12)
16            return true;
17
18        return false;
19    }
20 }
```

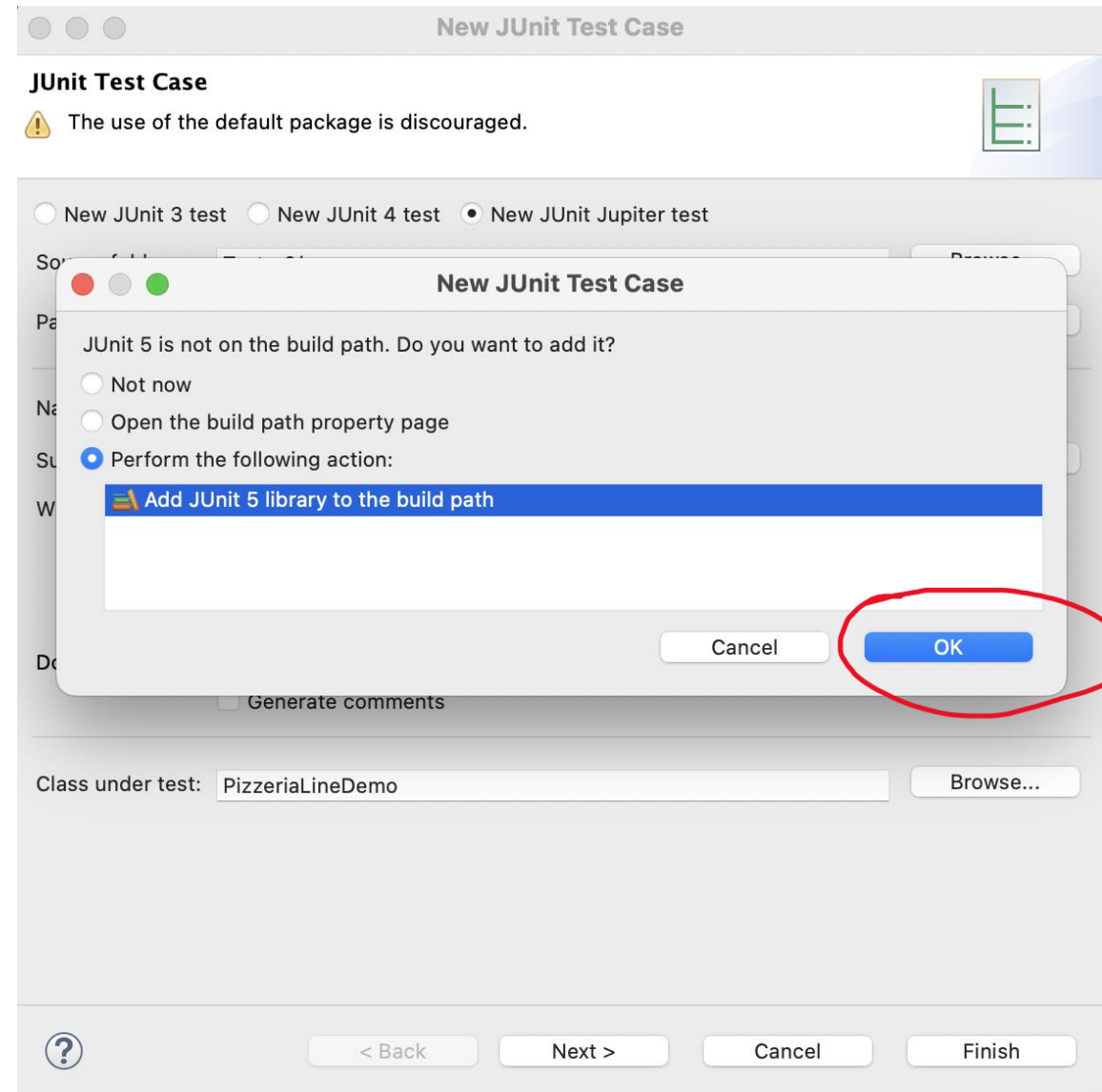
EvenCheckerTest (Dec 3, 2024 9:18:07 PM)

Element	Coverage
Tester4	81.0 %
src	81.0 %
(default package)	81.0 %
EvenChecker.java	68.6 %
EvenChecker	68.6 %
IsEven(int)	75.0 %
EvenCheckerTest.java	100.0 %



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

The screenshot shows the Eclipse IDE interface. The main editor displays the `Calculator.java` file with the following code:

```
1 public class Calculator {  
2  
3  
4     public static int add(int a, int b) {  
5         return a + b;  
6     }  
7  
8     public static int divide(int a, int b) {  
9         return a/b;  
10    }  
11 }
```

The `CalculatorTest.java` file is also open in the editor. The `Coverage` tab is active, showing a table of coverage data for the `CalculatorTest` run (Dec 3, 2024 8:34:29 PM).

Element	Coverage
Tester3	82.0 %
src	82.0 %
(default package)	82.0 %
CalculatorTest.java	84.6 %
Calculator.java	72.7 %

Calculator.java has 72.7% test coverage

(since we never instantiated a calculator object)