

COMP2511

23T1 Week 2

WEDNESDAY 1PM - 4PM (W13B)

FRIDAY 11AM - 2PM (F11A)

Slides by Alvin Cherk (z5311001)

Today

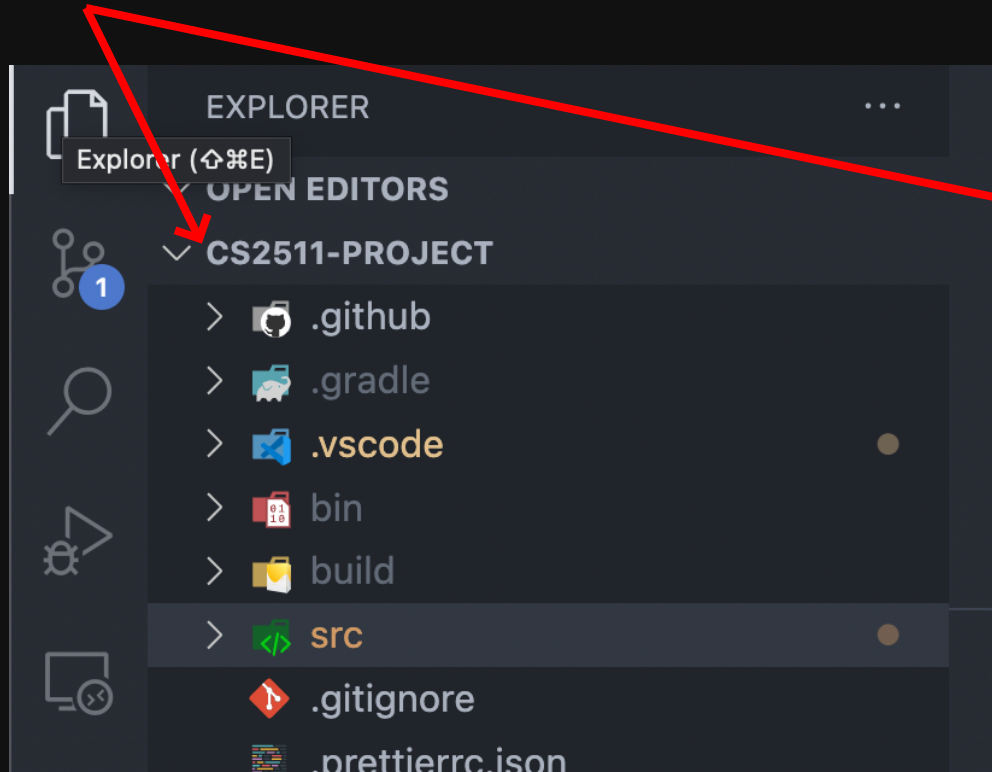
- Some VSCode, GitLab, Git tips
- Classes
- Commenting & Documentation
- Basic Inheritance
- Access Modifiers

Assignment I is out, please have a read of the spec and start early!

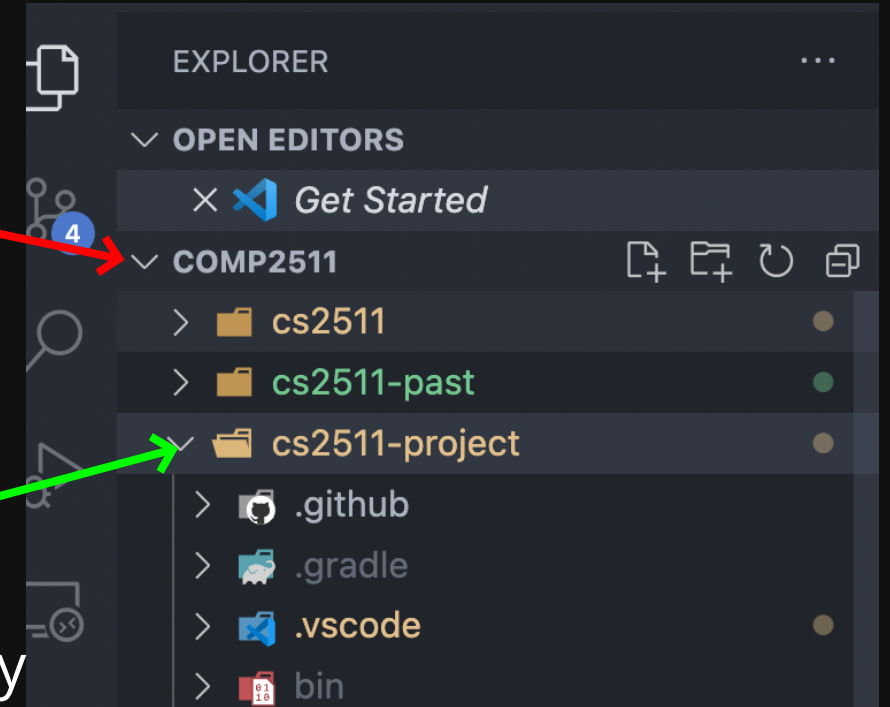
VSCode Config Tips

Ensure that you open the correct folder
example: cs2511-project

Name of folder open



Good



The folder I
want to actually
open

Bad

VSCode Config Tips

Ensure you have the correct Java extension installed

The screenshot shows the VS Code interface with the Extensions view open. The left sidebar lists several Java-related extensions. The main panel displays the details for the 'Extension Pack for Java' by Microsoft, which is currently in 'Preview' mode (v0.23.0). The extension is described as 'Popular extensions for Java development that provides Java IntelliSense, de...'. It has 13,120,043 downloads and a 4.5-star rating (49 reviews). The interface includes buttons for 'Disable', 'Uninstall', and 'Switch to Pre-Release Version'. Below the main details, there are tabs for 'Details', 'Feature Contributions', 'Changelog', and 'Runtime Status'. The 'Details' tab shows a list of included extensions: 'Language Support for Java(TM) ...' by Red Hat and 'Debugger for Java' by Microsoft. At the bottom, a description states: 'Extension Pack for Java is a collection of popular extensions that can help write, test and debug Java'.

EXTENSIONS: MARKETP...

java

- Extension Pack for J...** 45ms
Popular extensions for Java d...
Microsoft
- Maven for Java**
Manage Maven projects, exec...
Microsoft
- Debugger for Java** 9ms
A lightweight Java debugger f...
Microsoft
- Project Manager for ...** 12ms
Manage Java projects in Visua...
Microsoft
- Test Runner for Java** 42ms
Run and debug JUnit or TestN...
Microsoft
- Language Support fo...** 67ms
Java Linting, Intellisense, form...
Red Hat
- Spring Initializr...** 1.5M ★ 3.5
A lightweight extension based...
Microsoft
- Java Language Su...** 1M ★ 3
Java support using the Java C...
George Fraser
- Java Debugger** 533K ★ 3.5

Extension: Extension Pack for Java — cs2511-22t2-private

Extension Pack for Java v0.23.0 Preview
Microsoft | 13,120,043 | ★★★★★ (49)
Popular extensions for Java development that provides Java IntelliSense, de
Disable Uninstall Switch to Pre-Release Version
This extension is enabled globally.

Details Feature Contributions Changelog Runtime Status

Extension Pack (6)

- Language Support for Java(TM) ...** 67ms
Java Linting, Intellisense, formatting, refac...
Red Hat
- Debugger for Java** 9ms
A lightweight Java debugger for Visual Stu...
Microsoft

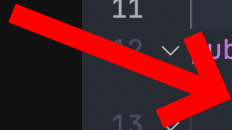
Extension Pack for Java

Extension Pack for Java is a collection of popular extensions that can help write, test and debug Java

VSCode Config Tips

Ensure that you are running code using the "Run" button

Always use this to
run code in this
course



```
You, last week | 1 author (You)
1 package example;
2
3 import java.util.Arrays;
4 import java.util.Scanner;
5
You, last week | 1 author (You)
6 /**
7  * Write a program that uses the `Scanner` class
8  * which reads in a line of numbers separated by spaces,
9  * and sums them.
10 */
11
You, last week • Week 1 ...
12 public class Sum {
13     public static void main(String[] args) {
14         /**
15          * new - Creates a new Scanner object. (Think of it like C Malloc, but Java's
16          * garbage collection frees it)
17          * Scanner is an object that allows us to specify an input
18          * System.in = stdin in C
19          */
20         Scanner scanner = new Scanner(System.in);
21     }
```


GitLab Tips

How to view pipeline output

COMP2511 > ... > repos > lab01

L

lab01 

Project ID: 261312 



☆ Star

0

🍴 Fork

667

🔗 8 Commits 🔗 2 Branches 🏷 0 Tags 📄 3.8 MB Files 💾 3.9 MB Storage

Forked from an inaccessible project

master



lab01 /

+



History

Find file

Web IDE



Clone



Update spec to match autotests for Splitter

Nick Patrikeos authored 2 days ago



794e0839



📁 Upload File

📄 README

📄 CI/CD configuration

⊕ Add LICENSE

⊕ Add CHANGELOG

⊕ Add CONTRIBUTING

⊕ Add Kubernetes cluster

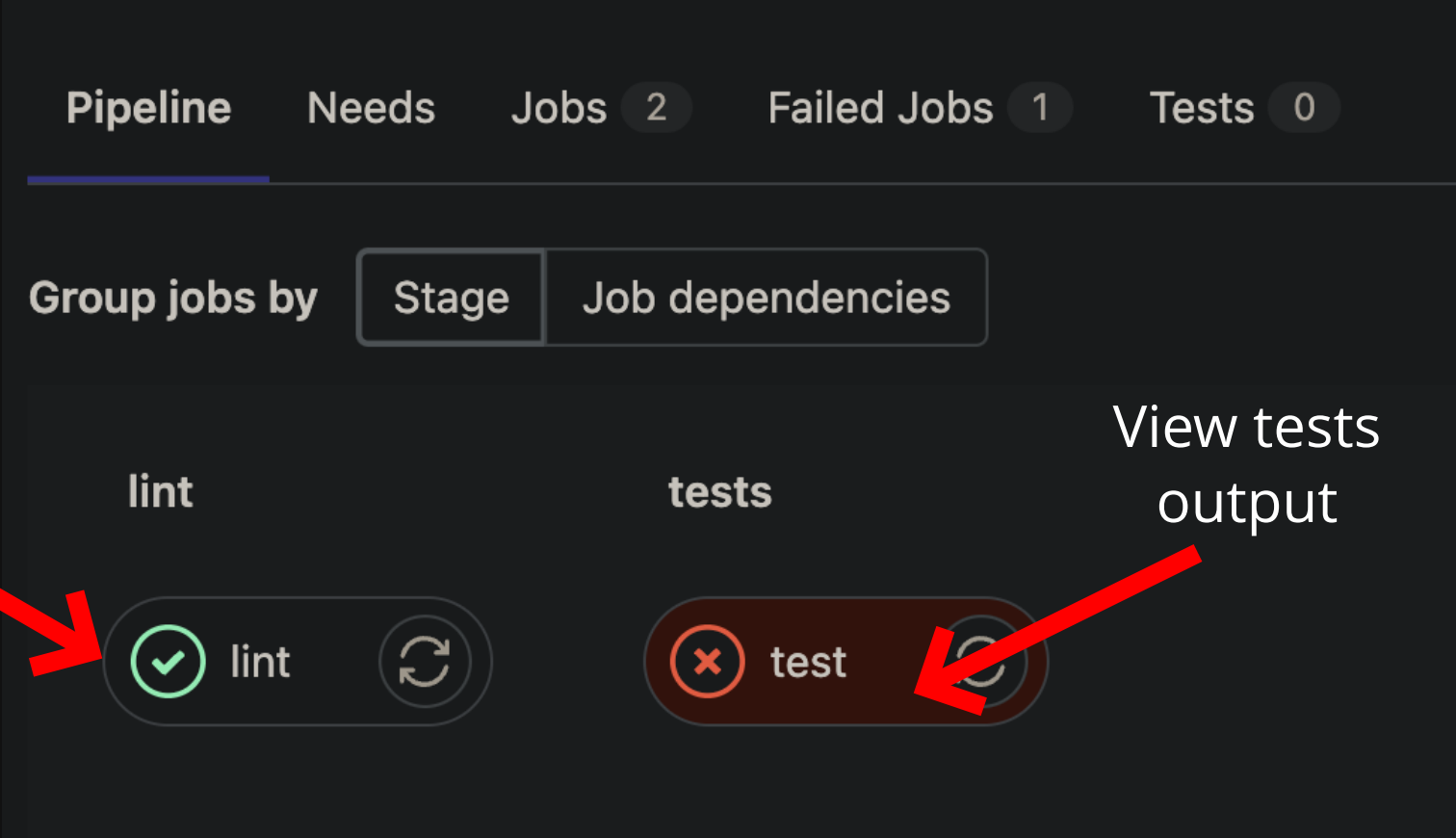
⚙ Configure Integrations

GitLab Tips

How to view pipeline output

View linting output

View tests output



The screenshot displays the GitLab CI/CD interface. At the top, there are tabs for 'Pipeline', 'Needs', 'Jobs' (with a count of 2), 'Failed Jobs' (with a count of 1), and 'Tests' (with a count of 0). Below these tabs, there are two buttons for 'Group jobs by': 'Stage' and 'Job dependencies'. The main content area shows two job cards. The first card is for the 'lint' job, which is green and has a checkmark icon. The second card is for the 'tests' job, which is red and has an 'x' icon. Red arrows point from the text 'View linting output' to the 'lint' job card and from 'View tests output' to the 'tests' job card.


lint tests

lint test

GitLab Tips

How to view pipeline output

```
18 Executing step_script stage of the job script
19 $ gradle test
20 Welcome to Gradle 7.4.2!
21 Here are the highlights of this release:
22 - Aggregated test and JaCoCo reports
23 - Marking additional test source directories as tests in IntelliJ
24 - Support for Adoptium JDKs in Java toolchains
25 For more details see https://docs.gradle.org/7.4.2/release-notes.html
26 Starting a Gradle Daemon (subsequent builds will be faster)
27 > Task :compileJava
28 > Task :processResources NO-SOURCE
29 > Task :classes
30 > Task :compileTestJava
31 > Task :processTestResources NO-SOURCE
32 > Task :testClasses
33 > Task :test
34 AverageTest > testAverage() FAILED
35     org.opentest4j.AssertionFailedError at AverageTest.java:18
36 SatelliteTest > testSatellite() FAILED
37     org.opentest4j.AssertionFailedError at SatelliteTest.java:18
38 SplitterTest > testSplitter() FAILED
39     org.opentest4j.AssertionFailedError at SplitterTest.java:28
40 3 tests completed, 3 failed
41 > Task :test FAILED
42 FAILURE: Build failed with an exception.
43 * What went wrong:
44 Execution failed for task ':test'.
45 > There were failing tests. See the report at: file:///builds/COMP2511/23T1/STAFF
    x.html
46 * Try:
```



Git Config

When you **commit** something, you are effectively saving the staged files as a new snapshot and **signing it** with your **name** and **email**. You have to configure your git identity if you haven't done it before.

```
1 git config --global user.name "Your Name Here"  
2 git config --global user.email "z555555@unsw.edu.au"
```

You then can add whatever email you have set in user.email on GitLab, so that it recognises all the commits that have been pushed to GitLab.

Please do this, its important Git etiquette.

It also allows your tutor to track the work you have done in your pair assignment.

Code Review

Code Review

- What is the difference between **super** and **this**?
 - **super** refers to the immediate parent class whereas **this** refers to the current class
- What about **super(...)** and **this(...)**?
 - **super()** acts as a parent class constructor and should be the first line in a child class constructor
 - **this()** acts as a current class constructor (can be used for method overloading)

```
1 package shapes;
2
3 public class Shape {
4     public String color;
5
6     public Shape(String color) {
7         System.out.println("Inside Shape constructor");
8         this.color = color;
9     }
10 }
11
12 public class Rectangle extends Shape {
13     public int height;
14     public int width;
15
16     public Rectangle(String color) {
17         super(color);
18         System.out.println("Inside Rectangle constructor with one argument");
19     }
20
21     public Rectangle(String name, int width, int height) {
22         this(name);
23         this.width = width;
24         this.height = height;
25         System.out.println("Inside Rectangle constructor with three arguments");
26     }
27
28     public static void main(String[] args) {
29         Rectangle r = new Rectangle("red", 10, 20);
30         System.out.println(r.color);
31         System.out.println(r.width);
32         System.out.println(r.height);
33     }
34 }
```

Code Review

- What is the difference between **super** and **this**?
 - **super** refers to the immediate parent class whereas **this** refers to the current class
- What about **super(...)** and **this(...)**?
 - **super()** acts as a parent class constructor and should be the first line in a child class constructor
 - **this()** acts as a current class constructor (can be used for method overloading)

```
1 package shapes;
2
3 public class Shape {
4     public String color;
5
6     public Shape(String color) {
7         System.out.println("Inside Shape constructor");
8         this.color = color;
9     }
10 }
11
12 public class Rectangle extends Shape {
13     public int height;
14     public int width;
15
16     public Rectangle(String color) {
17         super(color); // => Calling constructor of parent `Shape(String color)`
18         System.out.println("Inside Rectangle constructor with one argument");
19     }
20
21     public Rectangle(String name, int width, int height) {
22         this(name); // => Calling constructor `Rectangle(String color)`
23         this.width = width;
24         this.height = height;
25         System.out.println("Inside Rectangle constructor with three arguments");
26     }
27
28     public static void main(String[] args) {
29         // Rectangle(3 arguments) => Rectangle(1 argument) => Shape(1 argument)
30         Rectangle r = new Rectangle("red", 10, 20);
31         System.out.println(r.color);
32         System.out.println(r.width);
33         System.out.println(r.height);
34     }
35 }
```

Code Review

What are **static fields** and **methods**?

Static fields are variables that are common and available to all instances of a Class. They belong to the Class, rather than an instance.

Methods are a block of code that perform a task. You can think of them as functions of a class.

```
1 package circle;
2
3 public class Circle extends Object {
4     // Every class extends Object, it is not needed though
5     private static final double pi = 3.14159;
6     private int x, y;
7     private int r;
8
9     // Only 1 variable for all Circle objects
10    static int no_circles = 0;
11
12    public Circle() {
13        super(); // not needed
14        no_circles++;
15    }
16
17    public double circumference() {
18        return 2 * pi * r;
19    }
20 }
```

Documentation

JavaDoc

Documentation

- Why is documentation important? When should you use it
- What does the term "self-documenting" code mean?
 - **Code that documents itself.** It is readable inherently. Usually accomplished through variable name and function names
- When can comments be bad (code smell)?
 - Comments become **stale** & does not get updated with new changes
 - Possibly hinting that your design/code is **too complex**

Documentation

Single Line

```
1 // Single line comment
```

Multi-line comment

```
1 /**
2  * This is multi-line
3  * documentation
4  */
```

JavaDoc Documentation

```
1 /**
2  * Constructor used to create a file
3  * @param fileName the name of the file
4  * @param content contents of the file
5  */
```


JavaDoc

- JavaDoc is one way of documenting in Java.
- JavaDoc is a way of writing your comments
- It mainly targets class definitions and method/function definitions.
- In COMP2511, you will not have to use JavaDoc documentation unless asked. Though, it is a good idea to do it anyway in assignments.

JavaDoc

```
1  /**
2   * File class that stores content under a file name
3   */
4  public class File {
5      /**
6       * Constructor used to create a file
7       * @param fileName the name of the file
8       * @param content contents of the file
9       */
10     public File(String fileName, String content) {}
11
12     /**
13      * Constructor used to make a partial file when receiving a new file
14      * I.e., content.length() != fileSize with no compression
15      * @param fileName
16      * @param fileSize
17      */
18     protected File(String fileName, int fileSize) {}
19
20     /**
21      * Checks if transfer has been completed
22      * @return true if it has been completed
23      */
24     public boolean hasTransferBeenCompleted() {}
25 }
```

Inheritance

Inheritance

What is it?

In Java, a class can inherit attributes and methods from another class. The class that inherits the properties is known as the sub-class or the child class. The class from which the properties are inherited is known as the superclass or the parent class.

Known as a "is-a" relationship

Code Demo

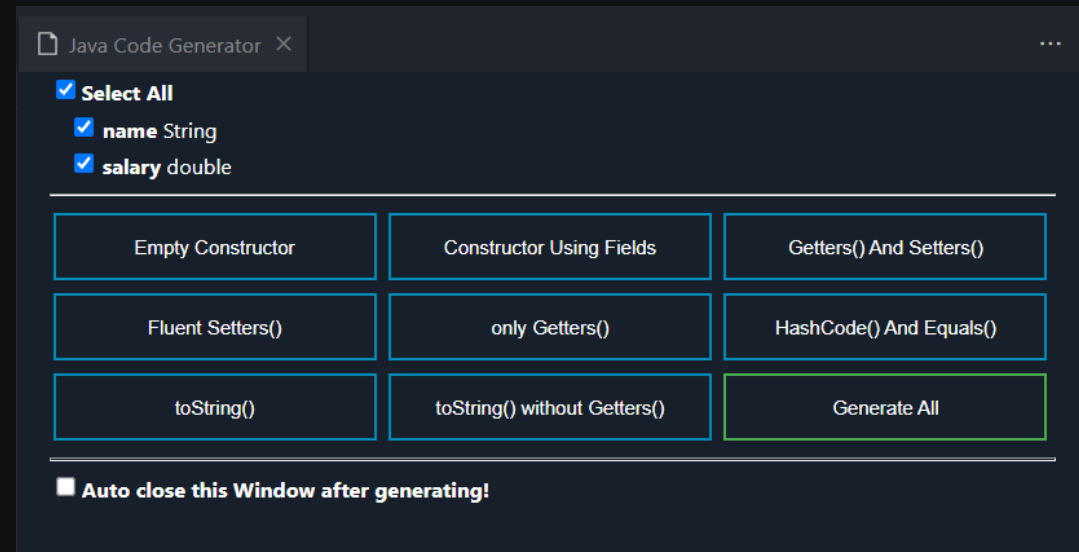
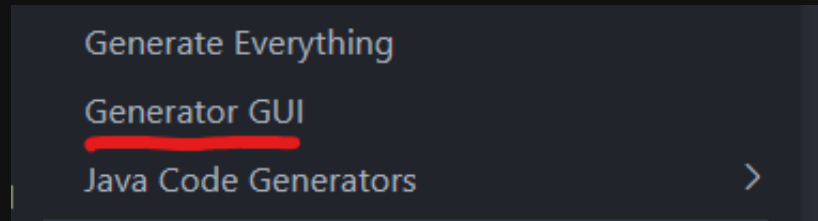
Employee.java & Manager.java

Code Demo

1. Create a **Employee** class with a **name** and **salary**
2. Create setters & getters with JavaDoc
3. Create a **Manager** class that inherits **Employee** with a **hireDate**
4. Override **toString()** method
5. Write **equals()** method

Code Demo

The java extension packs come with some features you can use to generate boilerplate code



Please **do not generate the equals method**. Write it yourself. This applies for Lab02 marking.

Code Demo

How many constructors does a class need?

Technically none. If a class is defined without a constructor, Java adds a default constructor

However, if a class needs attributes to be assigned (e.g., has a salary), then a constructor must be assigned.

If your class has attribute with no default values, then the **constructor must set these attributes**. You will lose marks in the assignment if this is not done. This is because variables with no values are dangerous, also constructor responsibility.

Code Demo

How many constructors does a class need?

```
1 import java.time.LocalDate;
2
3 public class BadConstructor {
4     public String name;
5     public double salary;
6     public LocalDate hireDate;
7
8     public BadConstructor() {
9         this.hireDate = LocalDate.now();
10        // salary and hireDate aren't assigned a value
11        // Technically, they're defaulted to null
12    }
13
14    public static void main(String[] args) {
15        BadConstructor e = new BadConstructor();
16        System.out.println(e.name);
17        System.out.println(e.salary);
18        System.out.println(e.hireDate);
19    }
20 }
```

Code Demo

How do you write a good equals method?

Since we are **overriding** an existing method (in the super most class called `Object`), we must follow the conditions described.

The conditions can be found in the [Java Docs](#)

equals

```
public boolean equals(Object obj)
```

Indicates whether some other object is "equal to" this one.

The equals method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any non-null reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in equals comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

The equals method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x == y` has the value `true`).

Note that it is generally necessary to override the `hashCode` method whenever this method is overridden, so as to maintain the general contract for the `hashCode` method, which states that equal objects must have equal hash codes.

Parameters:

`obj` - the reference object with which to compare.

Returns:

`true` if this object is the same as the `obj` argument; `false` otherwise.

The semantics of this was explored in a recent exam

Access Modifiers

Access Modifiers

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Private

It is accessible only to the same class (not including main). The most restrictive modifier.

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Public

It is accessible to everything. The least restrictive modifier.

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Protected

Can be accessed in the same package and in inheritance.

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Default

The default access modifier is also called **package-private**, which means that all members are visible within the same package but aren't accessible from other packages

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Feedback



<https://forms.gle/R4sMTTQzPC4vqXSN8>