**Java**

**Member**

A member of a class can be a:

* Variable
* Method

**Packages**

End of a domain name (.com, .co.uk), is the start of a package name.  
e.g. com.qa.packageName

**Auto-complete**

**Ctrl + Space triggers Eclipse’s autocomplete.**

Type “sysout” + Ctrl + Space to autocomplete to System.out.println().  
Used at the end of a function call, ctrl + space will show any expected params required.

**Scope**

The 3 types of scope are:

* Method
* Class
* Loop

Method declarations only exist within the method itself, and can only be accessed by the method.

Class scope declarations are accessible by any methods in the class and persist with the class or object.

Loop declarations only exist within the loop itself, and can only be accessed by the loop.

Curly braces are a good indicator of scope.

Scope has priority. From highest to lowest, it is loop, then method, then class. When looking for a variable, the code will look for it in a loop, then a method, then a class.

**Conditionals**

Difference between “&&/||” and “&/|”.

**If (False && True)** - && won’t check second condition because it knows it will evaluate to false overall.

**If (True || False)** - || Won’t check second condition because the whole expression already evaluates to True.

&& and || are more efficient than & and |, but sometimes you want to check both conditions regardless. So you would use & and | in that case.

**OOP**

**Encapsulation**

Where methods and classes own their own variables.

* Uses the **private**, **default**, and **protected** keywords in classes to secure data.
* Uses **public** methods, getters and setters, to access **private** data.

4 types of encapsulation with 4 types of access.

**Access Scope**

Class  
Package  
Subclass  
World

**Encapsulation**

Public – No scope restrictions.  
Default – Subclass and world variables cannot be accessed.  
Protected – Less restrictive than default. Only world scope is inaccessible.  
Private – Only things in the same class can be accessed.

**Polymorphism**

Two concepts: **Overloading** and **Overriding.**

**Overloading** is when you have two constructors or methods with the same name, that take a different number of parameters. The one that runs depends on the one that is called.

**Overriding** is when a subclass has the same method as a superclass. The subclass method runs, **overriding** the one in the superclass.

**Abstraction**

Abstract classes have abstract methods only. You cannot create an instance of an abstract class.

**Inheritance**

Where a subclass includes methods and variables from a superclass.

Java uses the **extends** keyword to denote inheritance.  
e.g. public class A **extends** B { }

**Interfaces**

Everything in an interface is abstract by default.

Declaration: public interface exampleInterface {}

Use keyword **implements** in a class declaration. e.g class A implements B {}

Useful when you want to use methods/variables from other classes, as you can implement multiple interfaces. Allows a class to act like it has multiple inheritance, which isn’t allowed in Java.

**Static**

When a method or variable is **static** it exists in a single location.

This is shared among all instances of the same class.

Static variables belong to a class, not an instance of that class.

To access something in a non-static way, create an instance of it.

Public class App {  
Main(){  
e.g. App app = new App();  
app.someMethod();  
 }  
}

**Garbage Collection**

Done automatically in Java.

“Balloon” objects are destroyed and then collected once all references to them are removed.

**Java Week 2**

**Debugging**

Errors cause code to stop execution and cause exceptions, bugs do not.

**Design Patterns**

Represent the best practices developed and used by experienced OO developers.

Usually solutions to common problems in development.

Can be thought of as templates for better code.

**Creational Pattern –** Creating objects while hiding the logic behind the creation.

**Structural Pattern –** Focussed on the way classes and objects are structured. Aims to use inheritance and interfaces to define the composition of objects for new functionality.

**Behavioural Pattern –** Focussed on the communication between objects.

**Java Database Connection**

**C – Create  
R – Read  
U – Update  
D – Delete**