Java

* Auto memory management
* Strongly typed
* Garbage collector frees things that no longer have references: set things you don’t plan to use again to null
* JDK has the libraries

Basic Programs

* Inside a class: public static void main(String[] args) { stuff }
* Everything inside classes
* Compile: javac HelloWorld.java
* Run: java (–classpath “mydirectory”) HelloWorld

Further Basics

* Class names are capitalised and the filename is the same.java
* class Child extends Parent { stuff }
* Child can overwrite Parent methods
* Overload is when you have the same function twice but different input
* Constructors have the same name as the class and happen on instantiation
* Parent constructor happens when Child is created
* Access: public is any; private is this class; default & protected is this package; protected is also subclasses outside

Next Section

* Object is an instance of a class: MyClass var = new MyClass();
* Object is the parent of every class, this gives the methods: getClass(); hashCode; toString();
* Methods are functions inside classes. Used by MyClass.myMethod();
* Static methods/variables are only made once each for all objects so most(all?) methods should be static
* Abstract classes have method declarations but not implementation. They can only be parent classes
* Import static allows the use of static members without class reference

Getters n Setters

* We use getVar() and setVar(var) inside classes to keep the private vars from being messed with as much

Strings

* String literals are constant, concat etc just make new strings. StringBuilder is an object for manipulatable strings
* The string pool lets identical string literals be recycled for efficiency, new String() prevents this
* There are a bunch of default String functions
* String.valueOf( ); converts numbers/bool/dates to strings
* Integer.parseInt( ); converts from string to integer, also available for Float and Double

Loops

* If else; switch case
* for(int c; c < 10; c++) ; for(element : array); do/while

Interfaces

* Contains abstract and default methods
* Classes using implementations are generally named with the suffix Impl
* public class MyClassImpl implements MyInterface, SecondInterface, Etc { }
* Abstract methods are like reminders of “required functions” that may differ in implementation in my understanding
* Default methods are functions that can be used but unlike classes you can have multiple interfaces
* Defaults can be @Override (ed)
* For conflicts you can choose eg interface A vs B: @Override public void method() {A.super.method();}
* You can even use both in the @Override or add more etc
* SuperClass > SuperInterface; Subtype > Supertype

Functional Interfaces

* Auto detected; only 1 abstract method
* Can be used with lambdas

Lambdas

* A lambda is an anonymous function that can be defined as a parameter
* Methods expecting functional interfaces can be given lambdas instead
* Ways of doing it:
* s -> System.out.println(s)
* System.out::println

Exceptions

* Exceptions propagate upwards until caught or they hit the top
* Checked exceptions must be caught or thrown further
* Methods that throw have that shown in the method
* public void exampleError() throws FileNotFoundException { }
* Checked exceptions are for errors that are expected in some cases (like file not found)
* Runtime exceptions don’t have to be catched explicitly like NullPointerException

Annotations

* Annotations such as @Override provide data
* They are interpreted by the compiler and as such can give warnings etc
* Im not sure if they have any code function

Optionals

* Optionals are used if the return value/parts of it may be null
* Created with Optional.of(thing); Optional.ofNullable(thing); Optional.empty();
* It has various functions to use/filter/parse the objects that it is created from
* If an element you try get() is null it throws NoSuchElementException. Use orElse/Get here

System Properties

* System.getProperty();
* Useful properties: path.separator line.separator user.dir user.home

Scheduled Tasks

* You can schedule tasks with java.util.Timer and java.util.TimerTask
* TimerTask has a run() function
* Timer sch = new Timer();
* sch.schedule(task, delay); happens once after delay
* sch.schedule(task, delay, repeatdelay); happens every repeatdelay after delay
* sch.cancel(); stops repeating (untested)

Streams

* java.util.stream.IntStream; Can be used to stream a sequence of Ints
* java.util.stream.Collectors; Can be used to accumulate/summarize elements
* list.stream() can be used to go through each element which then can be .filter() .map .collect etc

Maven

* Install maven (done)
* mvn –version
* mvn install installs dependancies locally so other projects can use them? -o can be used to stay offline
* mvn clean removes the old build
* -o offline; -fae fail at end; -ff fail fast; -fn fail never
* Maven wrappers are used for version control
* mvn –N io.takari:maven:wrapper –Dmaven={version}
* -N, --non-recursive is used for doing only the current project, no submodules
* Record the version for others to use as reference
* UNIX: ./mvnw clean package WINDOWS: mvnw.cmd clean package
* You can generate a project using maven in the terminal (eclipse can do it for you) details in txt file elsewhere
* mvn compile compiles Java sources
* mvn clean package creates the JAR file

Other

Stiff