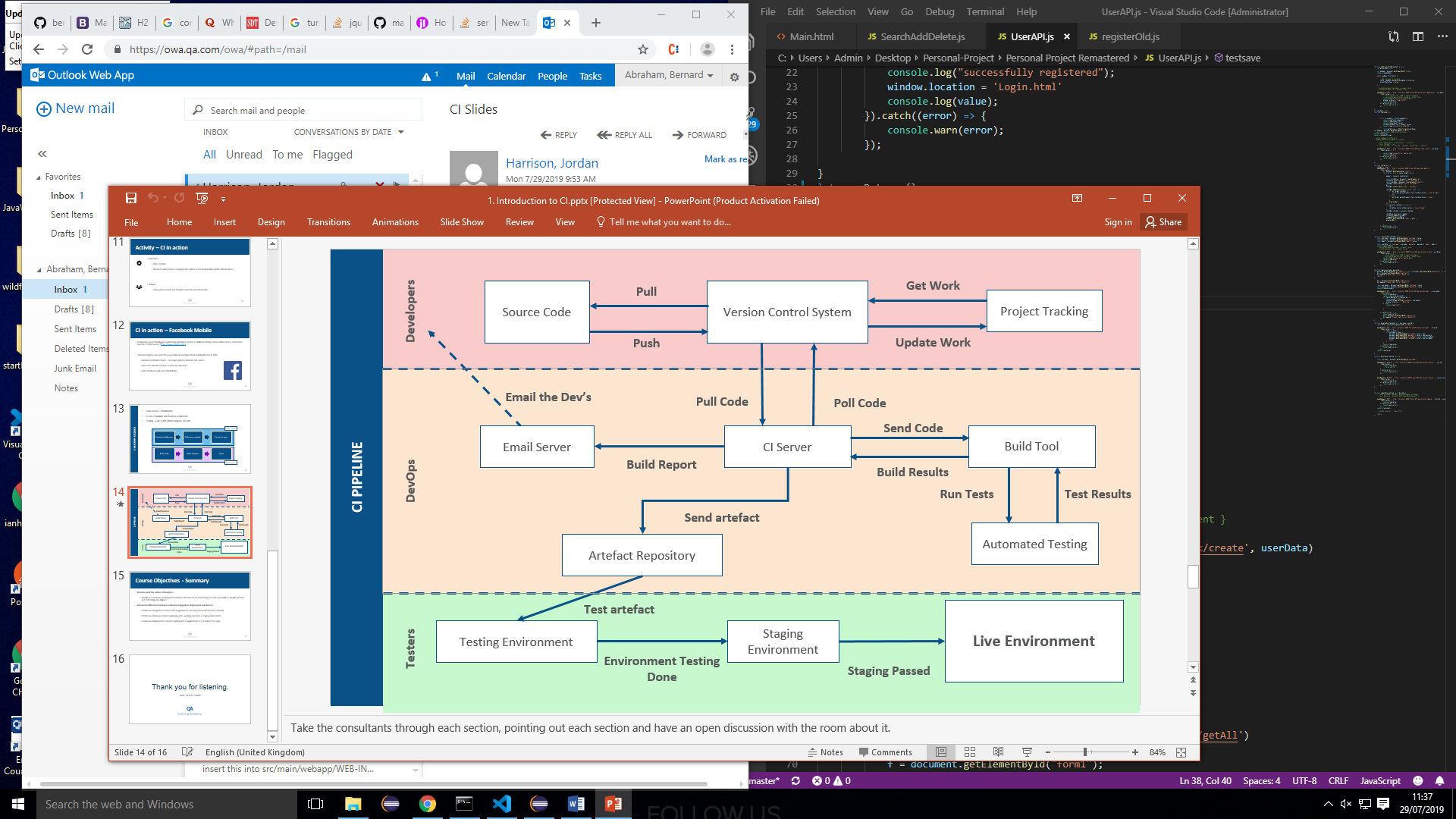
Week Eight – Monday - 29/07/2019

**Continuous Integration –** testing new features into software

**Continuous Delivery –** Deploying safe working code into staging environment

**Continuous Deployment –** Deployment of application to customer

Popular Interview Question:



Artefact repository – file storage(dropbox) for past builds/war files

Week Four – Monday - 01/07/2019

Primitive variables can’t be null, get set to zero by default if not given a value

Int[] nums = new{1,2,3,4};

***Array***List<> arrL = new Arraylist<>();

Can’t have mixed data types in Arrays, but you can have a list of objects

Linked lists are better for inserting items

numList.size for streams

filter – decreases values in stream by matching against a condition. Lambda expression goes in stream condition. .filter(x->x%2==0)

Reduce – reduces down to a single value .reduce((a,b)->a+b)

.Map(x->Maths.abs(x)) transforms each element into an element in a new list

forEach(system.out::println) similar to map but returns void

cohesion – how well your classes stick to a single purpose

**Encapsulation** – private variables public setters/getters. Public(global)/private(only read inside class)/default(visible to package)/protected (visible to child classes)

**Polymorphism** - Overriding (used in child class to change method) and overloading (changing parameters) defaultto shows class name

@override isn’t required but is similar to a comment

public String toString(){return ‘x’;} to override

**Inheritance** - child class can use parent classes attributes using ‘extends ClassName’

**Abstraction** – process of hiding away information through classes/methods/variables

Abstract classes as oppose to interfaces allow default methods/templates. You need to extend it to use it, you can’t create instances of it.

Interface uses implements keyword. You can implement as many interfaces but only extend one class.

In an interface every method is abstract. A class can’t inherit from an interface but an interface can inherit from another interface.

Casting – telling a variable what kind of object it is Animal a = new dolphin() the variable a is of type Animal

Instanceof to check the type of a variable. If (a instanceof Aardvark){return true;}

TDD – test driven development: Red, Green, refactor (rewrite code to be more efficient)

Try, catch, finally error exceptions. Errors are running out of memory whereas exceptions are less serious problems. Finally { s close()} for scanners is common.

Singleton design pattern is most common interview question. Makes only one instance of a class accessible, making a constant file in java which normally has no concept of this.

**SOLID** – Single responsibilities: each class does one thing

**Open/Close** -Method is open to extension but closed to modification so its reusable without having to go back to modify it

**L** –Liskov Sub Principle Child should be perfectly substitutable for the parent class. i.e flight should be implemented at the flightful level of birds who are able to fly

**I** – Interface segregation have more, smaller interfaces rather than one interface with many methods

**D** – Dependency inversion. Depend on abstractions (Abstract class/interface) not concretions (Classes that have method bodies that output something) Output should be the same type as the input

**Hash Maps** – Require two parameters, Map<keySet, valueSet> items = new HashMap<>();

You can’t put primitive variables inside the parameters. Maps are fast at finding items.

Getting an item by id can be done in two lines rather than making a stream that filters/gets

Set<entrySet<Long,Item>>

Week Three – Monday - 24/06/2019

<html>

<head>

<title> Website </title>

</head>

<body>

<h1><b> Name: Bernie </b> </h1>

<hr>

<h1><b><u> Occupation: Academytrainee</u> </b> </h1>

<h3><b> About me: Sandwhich eater </b> </h3>

<p> one two three four five six seven <input type="checkbox"> </p>

<footer>bye bye </footer>

</body>

</html>

**JavaScript -**  links html and CSS. Makes page interactive. React is a Javascript library for building user interfaces. Librarys are smallers bits of reusable code whereas frameworks provide larger skeletons for code.

Alerts stop the script from running until you click ok.

window.alert("hello");

in separate file

<script src="src.js">

Calls the file

Console.log are used for debug messages.

Window is the top level object for the browser.

Programming top criteria: rigor in expression, clearness in presentation, product adaptability/longevity, ease of scalability

Null is an explicit no return whereas undefined means impossible or inexplicit no return

NaN is toxic i.e it turns all added numbers to NaN (1 + Nan = Nan)

You don’t have to assign a type to a variable but you can with parseint() e.g for integers

You can use variables in any scope if theyre declared globally

Let y are used like normal variables and can’t be seen outside of their scope

Const are block-scoped and can’t have its value updated

Falsy values: false, null, undefined, “”, 0, Nan

You don’t have to define arguments for a function

**CSS –** bootstrap is a framework that can link to html files

External css is the most efficient. To the head, add <href=”css/style.css” rel=”stylesheet” type=”text/css”></link> to link to the style you’ve defined

**Selectors**

.className{….}

#id{….}

h1{….} !important takes the highest priority. Specificity and order also effect priority

H1 span {….}

**Bootstrap –** provides prebuilt styles.

Thursday 27/06/2019

**Promises –** have a request for a resolve or reject condition which outputs

**Javascript foreach** x.forEach((value, index, array)=>{})

returns void

**Week Two – Monday - 17/06/2019**

**JAVA Intro**

A variable is somewhere data is stored

Methods are functions that belong to a class

Public void yeet(){

} //public or private is the access modifier, void is the return type, means it doesn’t return anything

camelCase – used for variables, functions, no spaces in names, can’t start with number

PascalCase – use for classes

/\* \*/ for commenting out blocks or /\*\* \*\*/ for documentation

() surrounds parameters and arguments

.- used to access variables

Int everything after is an int until data type is changed

Return gives something back

Main method is where it starts

Methods are called or referenced in the main method

If a method isn’t void it has to return something

Int i = 12;

I++ //this increments the variable i by 1 each

Num += num; adds num to itself

Curly brackets you can’t access variables outside curly brackets unless its defined outside of them

Instance, local and loop scope

Static means “belongs to the class”. So it stays the same for every object in the class. The new keyword creates a new object. System.out.println() is static.

**Object** an object is an instance of a class,

***Instanceof*** compares an object to a type, returns True/False. Conditional **&&** is and **||**is or Boolean & and | or. Boolean version always runs both sides. Conditional one is most common.

Switch (day) { case 1: //gives a list of if/else statements they go on until something stops them

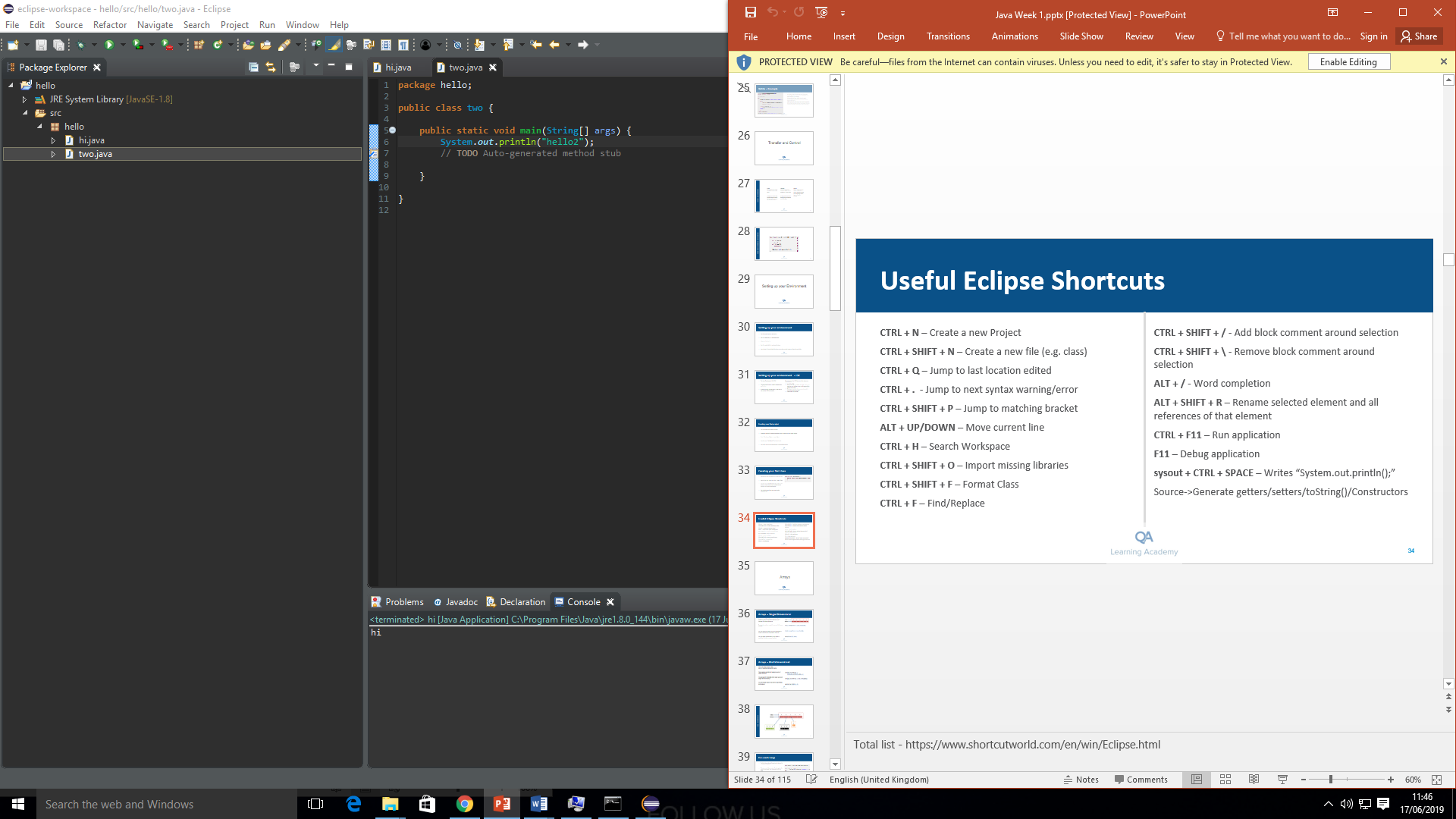
Break: kick out to the nearest set of curly brackets, you can break out of loops.

for (int = I; i<10; i++)

{system.out.println();}

Do while //this always runs atleast once, good for user input

Continue skips the rest of this iteration but it keeps looping Return ends the method full stop



If classes are in different packages they need to be imported

Methods are just functions in a class

New keyword makes it an object

Null means no reference to a value ie it doesn’t exist

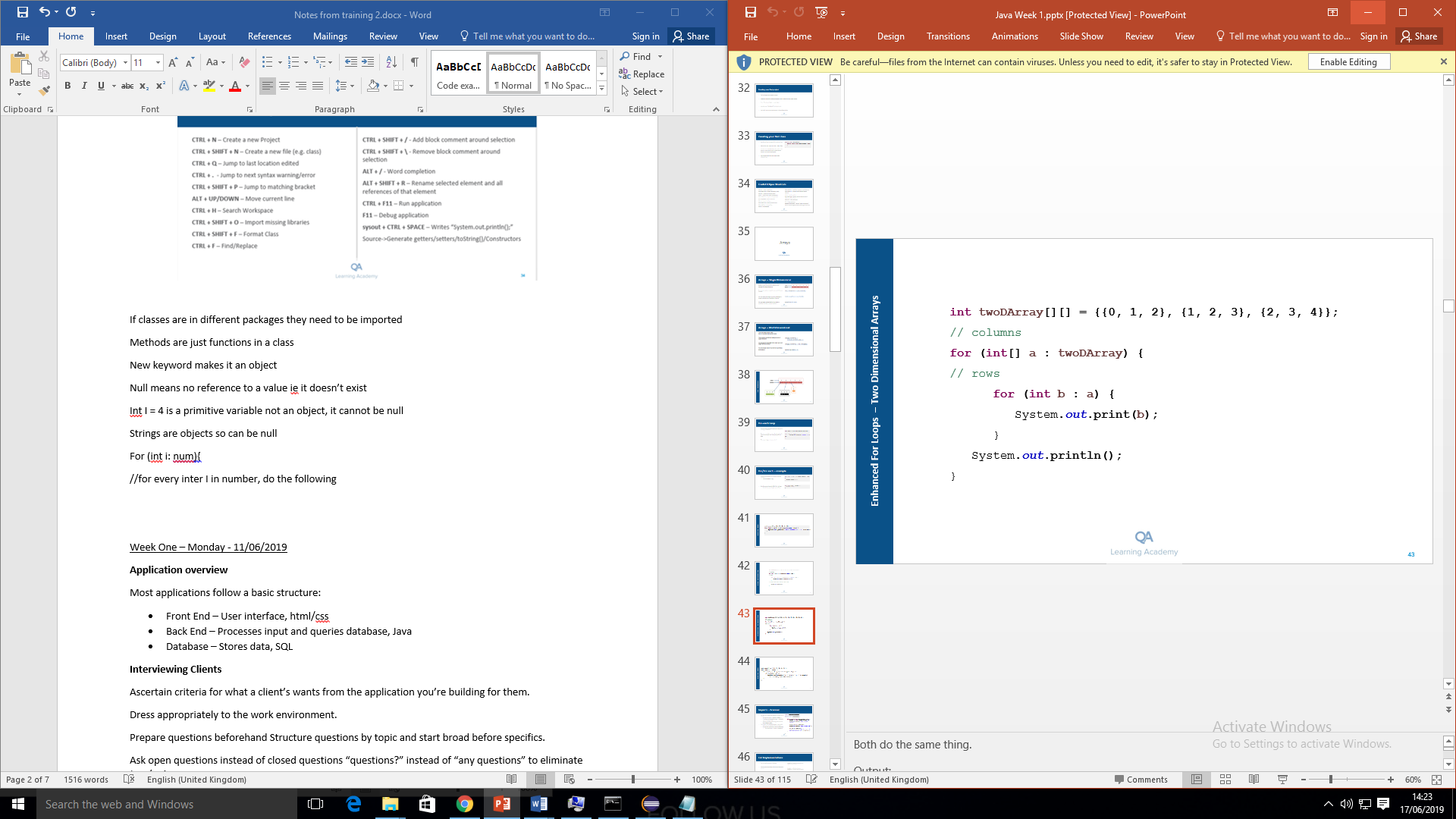
Int I = 4 is a primitive variable not an object, it cannot be null

Strings are objects so can be null

For (int i: num){

//for every inter I in number, do the following

Each data type has a capital letter equivalent that can be used for objects i.e Double, Int, String

enhanced for loop

**Scanners**

**Lists –** Arrays are static, if you want it bigger you have to recreate it but lists can be changed.

Array lists are indexed, linked lists only look at what’s in front and behind it, but searching is slower in linked lists as it has to look through the whole thing

List.add(4); //allows you at add a 4 to the end of the list

**OOP**

**Encapsulation** Private/public variables, getters and setters, data integrity

Least>most private: Public, protected, default, private

**Polymorphism** One method can have multiple forms through overriding and overloading

@overide is an annotation which tells you that method should be overriding something

**Inheritance** The ability to pass down attributes behaviours from parent classes

**Abstraction** Hiding away the implementation

Classes contain: Fields & methods or attributes & behaviours. Final classes can’t be extended, good for constants or file location.

Abstract methods have methods in them with no implementation

Super() means parent class, it is in a method automatically

Test driven dev: red>green>refactor

Refactor: reformatting the code so it does the same thing but more efficiently and is more maintainable

Maple project written in xml

Artefact ID – name of people who made it, Scope – part of the project it can see

**Interface** Nameusually ends in “able”. Interfaces are abstract meaning their methods have no implementation. “implements interface” written after “extends class”. Took flying out of birds and put it into an interface which is implemented on all birds that fly. Lists can only see the methods from that interface if you have a list of interfaces.

**Casting** allows you to change the methods a variable can access ((Dolphin) Bessie).echoLocate();

“treat this line like Bessie is a dolphin

**Garbage collection** Java automatically removes any variables that don’t point to anything from the memory.

Week One – Monday - 11/06/2019

**Application overview**

Most applications follow a basic structure:

* Front End – User interface, html/css
* Back End – Processes input and queries database, Java
* Database – Stores data, SQL

**Interviewing Clients**

Ascertain criteria for what a client’s wants from the application you’re building for them.

Dress appropriately to the work environment.

Prepare questions beforehand Structure questions by topic and start broad before specifics.

Ask open questions instead of closed questions “questions?” instead of “any questions” to eliminate ‘yes/no’ answers

Follow up with a summary or confirmation email

**Working in an Agile team**

Umbrella term for a set of programming principles, derivatives are leanOS, extreme programming, Kanban, safe, scrum

Uses development practices including peer programming, self-managing teams, empiricism, frequent feedback from client (typically every 2 weeks so you don’t go too far off course),

Daily Scrum – Brief daily stand up meetings at the start or end of the day

Two-week Sprint – One day of planning, 8 days of work, one day of review

Week One – Tuesday - 12/06/2019

**Introduction to SQL**

Three components of database: Tables (columns and rows), Schemas (layout/structure of the data) and Queries (retrieving data from tables).

Databases are either relational or non-relational. They are made up of tables, fields and records.

1 table> field (columns/categories) & records(rows/entries) inside fields

***cd “C:\Program Files\MySQL\MySQL Server 5.7\bin”***

# change directory to show cmd where to look for the executable, a bin folder is a folder of binarys as executables

***mysql -u root –p***

# this means run mysql with the username as root and a password

**Designing a database**

Data types: numbers, text and date

Numeric: BIT(1 to 64), BOOLEAN, TINYINT (-128 to 127), BIGINT, DECIMAL, FLOAT, DOUBLE. Using smaller more appropriate data types saves memory.

Float is similar to double but float is smaller

Date types: DATE, DATETIME, TIMESTAMP, TIME, YEAR (1901 to 2155). Early dates before 1901 can be stored as texts and converted with another programme.

Text: CHAR(fixed length at 0 to 50), VARCHAR(length of input),

You can save most things as a VARCHAR but it makes it harder to query them.

*Primary key*: a unique identifier

Constraints can be applied to tables: unique, not null, default, keys (unique and not null), auto-increment (every time a new row is added it adds 1 to the value, has to be used on primary key)

Reference a primary key in another table using a foreign key

*Entity relationship diagram (ERD)* – show design of database scheme with names, columns, relationships. Identify entity, attributes, primary keys, relationships, cardinality, draw a draft.

Entity type – table, Entity – a thing in the table

*Cardinality – how many times can a table appear in another table*

|| one only

O| zero to one

|< one to many

O< zero to many

If its ‘not null’ the relationship is not a ‘zero to’ one.

1)Identify entity customer, order, 2) attributes name, email, 3) primary keys ID, 4) relationships, 5) cardinality,

create database if not exists tesco\_DB;

use tesco\_DB;

*Data definition language:* Designing the schema of a table

3 key words *Create, Alter and Drop*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Create*** | ***Read*** | ***Update*** | ***Drop*** |  |  |
| ***DDL*** | Create | Show | Alter | Drop |  |  |
| ***DML*** | Insert | Select | Update | Delete |  |  |
|  |  |  |  |  |  |  |

**Creating tables**

*create database if not exists tesco\_DB;*

*use tesco\_DB;*

*drop table if exists Customer;*

*drop table if exists Customer\_Order*

*CREATE TABLE Customer(*

*Customer\_ID INT NOT NULL AUTO\_INCREMENT,*

*name VARCHAR (20) NOT NULL,*

*address VARCHAR (20) NOT NULL,*

*email VARCHAR (20) NOT NULL,*

*password VARCHAR (20) NOT NULL,*

*PRIMARY KEY (Customer\_ID)*

*);*

*CREATE TABLE Customer\_Order(*

*Order\_ID INT (10) NOT NULL,*

*Customer\_ID INT NOT NULL,*

*Date\_placed date NOT NULL,*

*FOREIGN KEY (Customer\_ID) REFERENCES Customer (Customer\_ID),*

*PRIMARY KEY (Order\_ID)*

*);*

**Misc**

Setting data to a primary key is the same as setting it to not null unique

An Orderline table is a join table which bridges a many to many table to make it into two one to many tables

Snake\_case is used in sql

**To amend a table:**

*ALTER TABLE Product ADD column username VARCHAR;*

*ALTER TABLE Product DROP column hi;*

*ALTER TABLE customer MODIFY COLUMN email VARCHAR (432);*

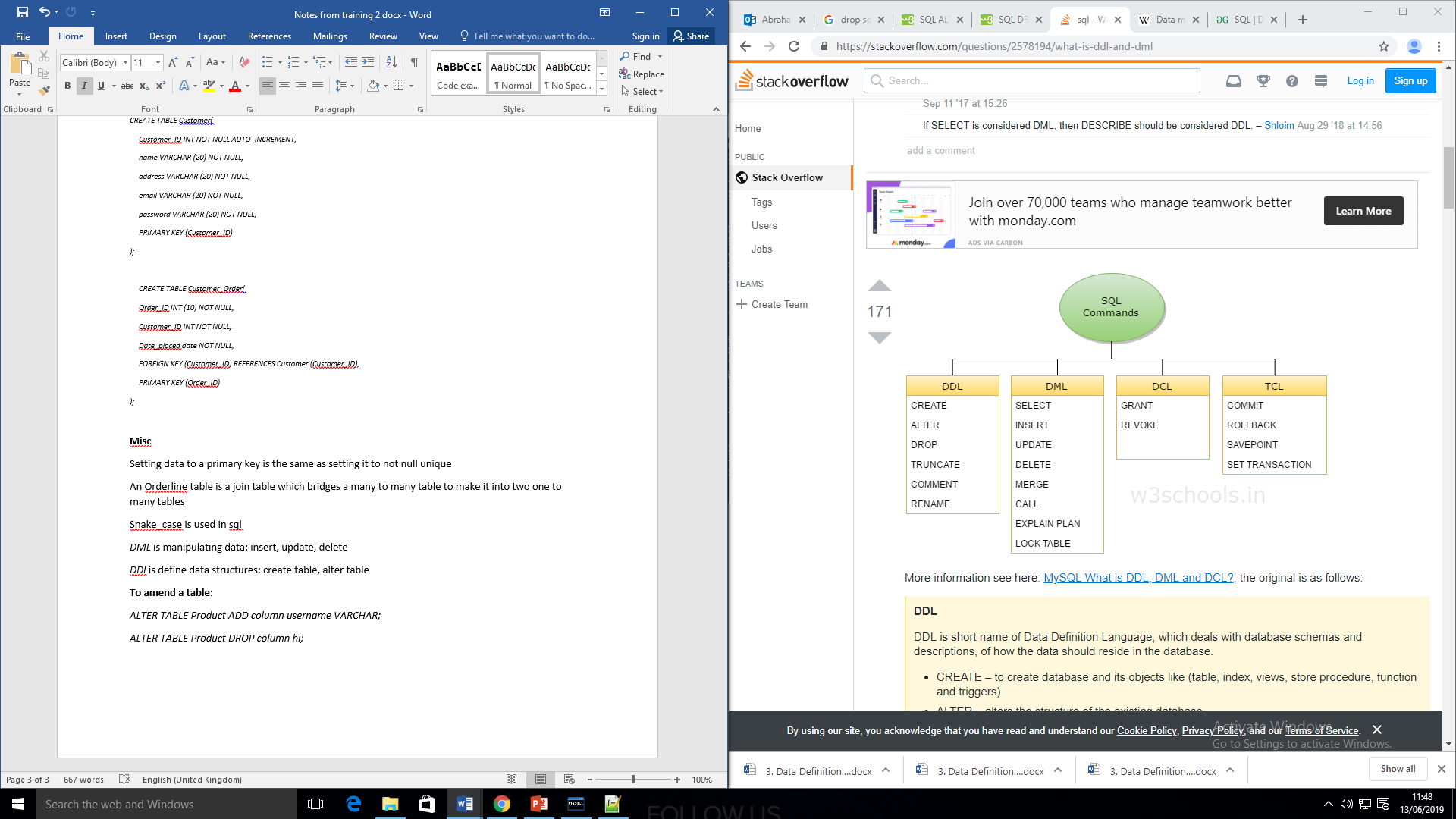
*DELETE FROM customer WHERE customer\_name = ‘Steve Jones’;*

*#where field specifies the column you are targeting*

**DML & DDL**

*DML* is manipulating data: insert, update, delete

*DDl* is define data structures: create table, alter table



DML:

INSERT INTO table\_name (

Column1, column2, column3)

VALUES(

Value1, value2, value3);

*UPDATE customer SET age=22, email='Steve@gmail.com' WHERE name="Steve Bills";*

*Composite key:* made up of 2 other keys*.* The individual keys aren’t unique but the combination of them is. Syntax: PRIMARY KEY (C-ID, P-ID),

**Select Keyword**

The \* selects the whole table

Select email, name from customer; #selects specific fields from table\_name;

Select distinct field\_name from table\_name; #lists the unique values of the field

**Where clause – operators**

= equals, >, <, !, between(within an exclusive range),

like(search for a pattern)

like “%a” #something ending a, like”\_a\_” #three letter word with a in middle, like”%jake%” #any word with jake in it

in(specify multiple possible values for a column), is null (select everything where field is null)

select \* from customer WHERE last\_name like "s%";

select \* from customer WHERE city in (“Birmingham”, “liverpool”);

Alias: select firstname as name from customer. #since you are creating a temporary table with selecting, this allows you to rename the output. Useful because it stores your result.

Select customer\_name, age from customer order by age; # this selects two values from customer and orders them by age.

Select \* from customer order by age limit 1; #limit allows you to take the first x results

Order by is ascending by default, for descending its order by age DESC

**Aggregates and joins**

Aggregate is a function you run a field through that returns one record

***Count*** counts the amount of value in that field

***Sum*** *sum total,* min gets minimum value, AVG gets average field]

As is an alias

**Group by** select customer\_id, MAX(price) as maxP FROM orders GROUP BY customer\_id WHERE rating =5;

Group by allows you to group together all records of a particular values

Finds the most expensive order that was rated 5 stars

Order by sorts by ascending or descending order

Nested query (a query within a query)

SELECT product, price FROM products WHERE customer\_id IN

(SELECT customer\_id FROM customer WHERE age>50);

Uses two tables product and customer

Give product and price of customer over 50

Creates temporary table and searches it

**Joins**

Allows you to join two tables using a common field (inner join)

Select \* from table1 t1 JOIN table2 t2 ON t1.field=t2.field;

Same data type

One will be primary key and one will be foreign key

This does an inner join that selects common values and joins them

Aliasing allows you to shorten names

Left and right joins

Customer LEFT OUTER JOIN customer order

Takes every value from table on left from primary key and matches it to table on right

Sql is read through twice so aliases can be stated before they’re defined

**Views**

Views are similar to variables in that they