## Explain differences between Java and JavaScript

Java is a compiled language whereas JavaScript is a scripted language. This means that all Java code has to go through a compiler before it can be run. This checks for all checked exceptions that would otherwise be thrown during runtime in a Javascript application, all unchecked exceptions works the same though. This also makes it easier to write the actual code you want to write in Java, as if you misspell one of your variables it can’t be used later, whereas JavaScript just creates a new variable on the global scope.

Java code is written through an Intergraded Development Environment (IDE), which turns the code into bytecode. This bytecode is not readable by the human eye and needs a Java Virtual Machine (JVM) to run it. JavaScript is executed in the same language that it’s written by a JavaScript engine. The compiler turns Java code into bytecode and then a JVM turns the bytecode into the native code of your computer.

There is some similarities between Java and JavaScript but in general, they are as different as apples and cats. Both are Object Oriented Languages, and both needs an interpreter to run. Java uses a JVM, and JavaScript uses a browsers JavaScript engine.

Code completion is a lot better in Java.

JavaScript is more loosely typed than Java. No type checking as an example.

## Explain the following features in JavaScript

### The use of: ”use strict”

As mentioned earlier JavaScript is a loosely typed language, so type checking isn’t needed. If “use strict” directive is added to JavaScript, code the code is then defined to be executed in strict mode. This makes previously accepted bad syntax into errors. When writing in strict mode it’s not possible to make undeclared variables, and it’s no longer possible to add a number to a String. Things not allowed in JavaScript when “use Strict” is used:

* Using a variable or an object without declaring it.
* Deleting a variable or a function
* Duplicating parameter names
* Octal numeric literals and escape characters
* Writing to a read only property
* The string eval and arguments can’t be used as a variable
* The with statement

The “use Strict” directive is only recognized at the beginning of a script or function.

### Variable/function-hoisting

In JavaScript all variable declarations are moved to the top of a given scope, this is called variable hoisting. This means that it’s possible to use a variable after it’s been declared. Due to this behavior it’s recommended to move all variable declarations to the top of the scope you’re working in as it’s only the declarations that are moved to the top, not the initialization.

When a function is written both the functions name, and the function definition is hoisted to the top of a given scope. This is called function hoisting. This only happens for function declarations and not for function expression. There it’s only the name of the function expression that’s hoisted.

### *This* in Javascript and differences from Java.net

### Immediately-Invoked Function Expressions (IIFE)

IIFE is when a function is executing itself through the (function(){})() expression. This can be used to avoid variable hoisting within blocks.

### Use the Debugger to explain basic things all objects inherits from object

### User defined Callback Functions

In JavaScript functions are first-class objects, so they can be used in a first-class manner just like any other object (String, Number, etc.) as they are objects themselves. This means that functions can be stored in variables, passed as arguments, created within other functions and be returned from functions.

Passing a function as a parameter into another function is the essence of a callback function. When a callback has been parsed as a parameter, it can then be executed inside the other function. The callback we parse is not executed immediately but is instead “called back” at some specified point inside the containing functions body.

It’s possible both to use named or anonymous functions as callbacks. A lot of the time an anonymous functions is going to be fine, but if we end up in a chain of callbacks, it’s often a good idea to name the functions and then parse them into the function as a callback, to make reading of the code easier.