Kotlin

* Can run anywhere where JVM can run
* Statically typed – the type of every variable is known by a compile time; it doesn’t figure out the types at runtime
* Object oriented
* However it can use Functional programming – you can use functions as values, store them in variables, return them from functions and pass them as arguments to functions, also you can declare immutable object; functional programming allows Kotlin to be concise(кратък, сбит, стегнат) which means you have to write less code to accomplish sth than in Java
* Guiding principles:
* Conciseness
* Safety – null pointer exceptions
* Pragmatism – it is not a research language, Kotlin is designed to solve real world problems; there is always more than one way to do sth in Kotlin; in Kotlin world no way is considered better than the other
* Interoperability – Kotlin was designed to be interoperable with Java

-No need to declare exceptions as in Java; ‘throws’ keyword doesn’t exist in Kotlin

- The Ternary operator is gone and replaced by the if expression

- The for loop doesn’t exist in Kotlin – e.g -> for(int i=0; i<20; i++)

- Kotlin doesn’t have a ‘static’ keyword – replaced with top-level functions, the concept is still there but not visible; the Kotlin code is still being compiled down to java bytecode when targeting the JVM

- No ‘new’ keyword when creating a new object of a class

- !!! Main difference here in Kotlin is the ‘==’ operator:

\* in Kotlin the ‘==’ operator checks for structural equality (while in Java it is only checking for referential equality and ‘.equals’ checks for structural equality) when it is comparing instances of classes; actually refers to the ‘.equals’ operator; it is also a safe operator and won’t give a null pointer expetion

\* in Kotlin the ‘===’ operator checks for a referential equality

-a function that returns Unit is like a void function in Java, however here a unit function actually returns a singleton unit instance

- ?: Elvis operator; lets you assign a default value when an expression evaluates to null

- safe call and safe cast operators used to ignore Null Pointer Exceptions; meaning that the app won’t crash

- with a safe call (?) we never get a Null Pointer Exception, however with (!!) operator we can get an exception and address the exception

-the default visibility is public; you can have as many public classes in one file as you want; here you can declare also a private class which means that everything in the same file will have an access to that class