**1**

**KOTLIN**

* Kotlin is a statically typed, general-purpose programming language developed by JetBrains.
* That has built world-class IDEs like IntelliJ IDEA, PhpStorm, Appcode , etc.
* It was first introduced by JetBrains in 2011 and a new language for the JVM.
* It is Object-oriented language, and a “better language”than Java, but still be fully interoperable with Java code.
* It is sponsored by Google, announced as one of the official language for Android Development in 2017.
* Combines object-oriented and functional programming features.

**Why Kotlin ?**

* Concise (short) : It is more concise than Java and you would need to write approximately 40% fewer lines of code when compared to Java.
* Interoperability: This programming language is highly interoperable with Java, You will never face any difficulty using Kotlin in a Java project.

**2**

* Feature-rich: It provides several advanced features such as Operator overloading, Lambda expressions, String templates,etc.
* Easy : It is easy to learn programming language. If you have come from a Java background, you would find it easy to learn Kotlin.
* Less error : prone: As mentioned before, Kotlin is a statically-typed programming language,which makes you able to catch errors at compile - time, as Statically typed programming languages do type checking at compile-time.
* Safe : It provides the safety from most annoying and irritating NullPointerExceptions by supporting nullability as part of its system. Every variable in Kotlin is non-null by default.
* Smart Cast : It explicity typecasts the immutable values and inserts the value in its safe cast automatically.

**Where Kotlin is used?**

* You can use Kotlin to build Android Application.
* Kotlin can also compile to JavaScript, and making it available for the frontend.

**3**

* It is also designed to work well for web development and server-side development.

**Classes and Object**

* A class is a blueprint, and an object is an instance of a class.
* Usually, we define a class and then create multiple instances of that class by creating Objects.

**Constructors**

* A constructor is a special member function that is invoked when an object of the class is created primarily to initialize variables or properties.
* A class needs to have a constructor and if we do not declare a constructor, the the compiler generates a default constructor.

**Kotlin has two types of constructors—**

**Primary constructor**

**Secondary/Custom Constructor**

**Ex- Primary**

Class Sum constructor(val a: Int, val b :Int){ //code

}

**4**

Class Sum (val a : Int, val b:Int){

//code

}

**Init Block**

* The Primary constructor cannot contain any code, the initialization code can be placed in a separate initialize block prefixed with the inti keyword.

Class Sum{

//code

init{

Println(“Init Called”)

}

}

**Custom Constructors(Secondary Constructor)**

* A class in Kotlin can have at most one primary constructor, and one or more custom/secondary constructors.

**5**

* The primary constructor initializes the class, while the secondary constructor is used to initialize the class and introduce some extra logic.

Constructor(a : Int){

Println(“Result $a”)

}

Constructor(a : Int, b : Int){

Println(“Result ${a+b}”)

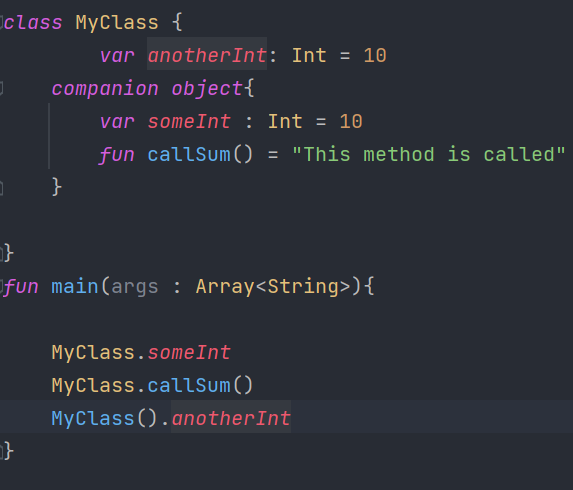
}

**Companion Object**

* In Kotlin or any other programming language like Java and C#.
* Whenever we want to call the method or whenever we want to access the members of a class then we make the object of the class and with the help of that object, we access the members of the class.

**6**

**Example:**



**Inheritance**

* It is the mechanism by which one class is allow to inherit the features(fields and methods) of another class.
* One object acquires all the properties and behaviors of a parent object.
* It is an important part of OOPs (Object Oriented programming system).
* Subclass can reuse methods and fields of the Parent class.
* By default, Kotlin classes are final-they can’t be inherited. To make a class inheritable, mark it with the open keyword.

**7**

**this vs super Keywords**

* The this keyword points to a reference of the current class, while the super keyword points to a reference of the parent class.
* this can be used to access variables and methods of the current class, and super can be used to access variables and methods of the parent class from the subclass.

**Interface**

* An interface is a reference type.
* It is similar to class.
* It is a collection of abstract methods.
* A class implements an interface, thereby inheriting the abstract methods of the interface.
* Only method signature ,no body
* Interfaces specify what a class must do and not how It is the blueprint of the class.

**Abstraction**

* Abstraction is one of the core concept of Objected Oriented Programming.

**8**

* When there is a scenario that you are aware of what functionalities a class should have.
* But not aware of how the functionality is implemented.
* If the functionality could be implemented in several ways, it is advised to use abstraction.
* It contain adstract on non-abstract : variables and functions.
* Abstract Class cannot be initiated but could be extend class a Super Class.

**Enum**

* In programming, sometimes there arises a need for a type to have only certain values.
* To accomplish this, the concept of ecumeration was introduced. Enumeration is a named list of constants.
* In Kotlin, like many other programming languages, an enum has its own specialized type, indicationg that has a number of possible values.
* Enum constants aren’t just mere collections of constants-these have properties, method etc.

**9**

* Each of the enum constants acts as separate instance of the class and separated by commas.
* Enums increase readability of your code by assigning pre-defined names to constants.

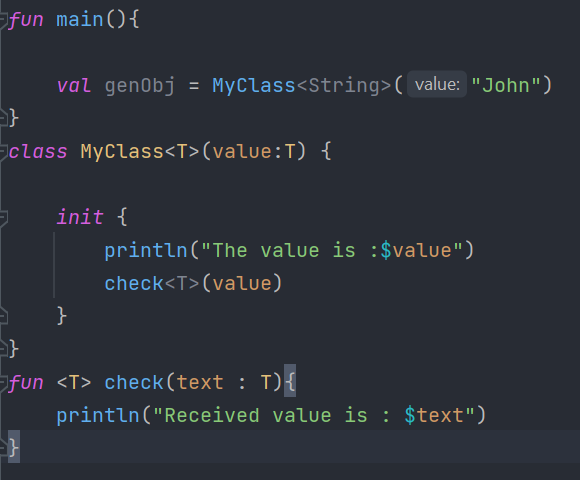
**Generics**

* Generic is defined as a product without a brand name.
* The definition of generic is something without a brand name.
* An example of generic is the type of soap with a store’s label that says “soap”, but without a brand name.
* Generics are the powerful features that allow us to define classes, methods and properties.
* Which are accessible using different data types while keeping a check of the compile-time type safety.
* A generic type is a class or method that is parameterized over types.
* We always use angle brackets <> to specify the type parameter in the program.

**10**

**Advantages of Generics**

* Type casting is evitable-No need to typecast the object.
* Type safety-Generic allows only single type of object at a time.
* Compile time safety-Generics code is checked at compile time for the parameterized type so that it avoids run time error.

****

**Lambdas**

* It is just a Function without name.
* Also known as Anonymous Functions.

**11**

Lambdas Expression

Parameters {x,y -> x+y} Body

{x -> x \*x}

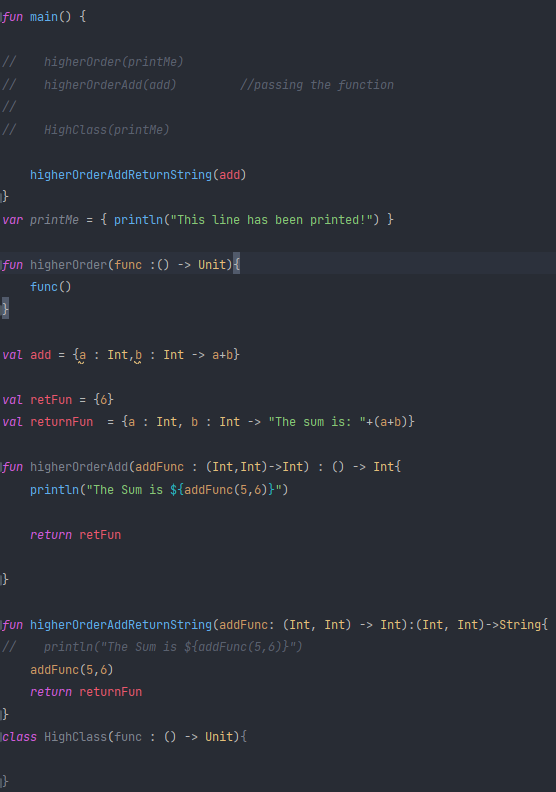
{x,y,z -> x+y+z}

{str -> println(str)}

**High Order Functions**

* A Function which can accepts a function as parameter or can returns a function is called Higher-Order function.
* Kotlin functions can be stored in variables and data structures, passed as arguments to and returned from other higher-order functions.

**12**

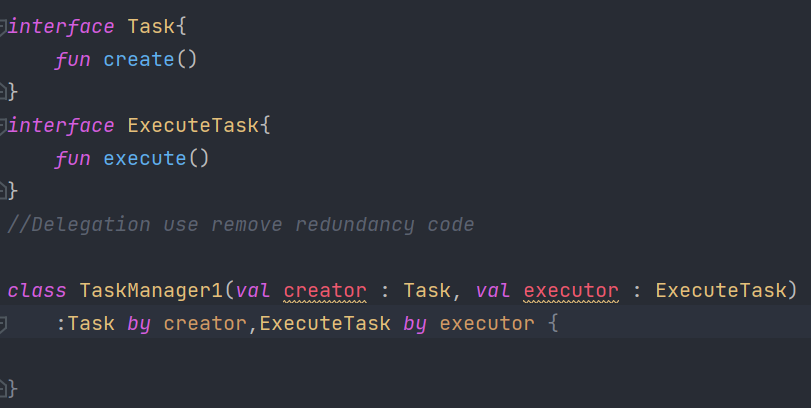


**Delegations**

* Inheritance implementation in classes and functions can be altered with the help of delegation techniques.
* Object-oriented programming languages support it innately without any boilerplate code.
* Delegation is used in Kotlin with the help of “by ” keyword.

**13**

**Example of Delegations**

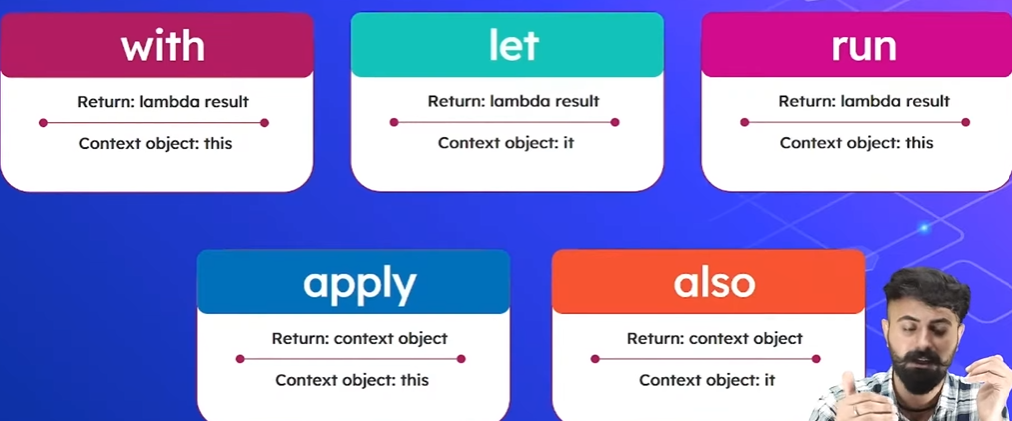


**Scope Functions**

* The Kotlin standard library contains several functions whose sole purpose is to execute a block of code within the context of an object.
* When you call such a function on an object with a lambda expression provided, it forms a temporary scope.
* In this scope, you can access the object without its name.

**14**

**Scope Functions**



**Collections**

* A collection usually contains a number of objects (this number may also be zero) of the same type.
* Objects in a collection are called elements or items.
* For example, all the students in a department form a collection that can be used to calculate their average age.

**The Following collection types are relevant for Kotlin:**

* **List**

1. It is an ordered collection with access to elements by indices-
2. Elements can occur more than once in a list.
3. An Example of a list is a sentence: it’s a group of words, their order is important, and they can repeat.

**15**

* **Set**

1. It is a collection of Unique elements.
2. It reflects the mathematical abstraction of set : a group of objects without repetitions.
3. Generally, the order of set elements has no significance for example, an alphabet is a set of letters.

* **Map**

1. It (or dictionary) is a set of key-value pairs.
2. Keys are unique, and each of them maps to exactly one value.
3. The values can be duplicates.
4. Maps are useful for storing logical connections between objects, for example, an employee’s ID and their position.

**Nullability**