```kotlin
import java.io.InputStream
import java.io.OutputStream

// Simplest way to declare class
class Deposit {}

// example usage of 'constructor' keyword, and making 'instance'.
// Declare class Person1
class Person1 constructor(val firstName: String, val lastName: String, val age: Int?) {
    /* code */
}
// Make Instance of Person1
val person1 = Person1("Alex", "Smith", 29) // No need to say 'new' keyword!

// example usage of 'init', 'require', secondary constructor
// Declare class Person2
class Person2( val firstName: String, val lastName: String, val age: Int? ) { // 'constructor' keyword is optional for primary constructor.
    // Initialize
    init {
        // Check input values via standard library function 'require'
        require(firstName.trim().length > 0) { "Invalid firstName argument." }
        require(lastName.trim().length > 0) { "Invalid lastName argument." }
        if (age != null) {
            require(age >= 0 && age < 150) { "Invalid age argument." }
        }
    }
    // secondary constructor
    constructor(firstName: String, lastName: String) : this(firstName, lastName, null) {}
}
// Make Instance of Person2
val person3 = Person2("Inha", "Woo") // age will be null

// Nested Class
class Outer {
    private var privateInt = 3
    class staticNestedClass { // Equivalent to static nested class in Java
        init {
            // println("Outer's static nested class : ${privateInt}") // ERROR! Cannot access private member.
        }
    }
    inner class innerNestedClass { // Equivalent to non-static(inner) nested class in Java
        init {
            println("Outer's inner nested class : ${privateInt}") // OK
        }
    }
}

// this@label
class A {
    var myVal = 1
    inner class B {
        var myVal = 2
        init {
            println("Field <myVal> from B: " + this.myVal) // 2
            println("Field <myVal> from B: " + this@B.myVal) // 2
            println("Field <myVal> from A: " + this@A.myVal) // 1
        }
    }
}

// enum class
enum class ecPlatformId {STD5, STD5W, PRM5, CCIC, CCNC, CCIC27, TCI}
enum class Planet(val mass: Double, val radius: Double) {
    MERCURY(3.303e+23, 2.4397e6),
    VENUS(4.869e+24, 6.0518e6),
    EARTH(5.976e+24, 6.37814e6),
    MARS(6.421e+23, 3.3972e6),
    JUPITER(1.9e+27, 7.1492e7),
    SATURN(5.688e+26, 6.0268e7),
    URANUS(8.686e+25, 2.5559e7),
    NEPTUNE(1.024e+26, 2.4746e7)
}

// Singleton class = 'object' in Kotlin
object myButton {
    private var count = 0
    fun press() {
        println("Calling myButton.press() : ${++count}")
    }
}

// interface
interface Document {
    val version: Long
    val size: Long
    val name: String
        get() = "NoName"
    fun save(input: InputStream) // import java.io.InputStream
    fun load(stream: OutputStream) // import java.io.OutputStream
    fun getDescription(): String {
        return "Document $name has $size byte(-s)"
    }
}
class DocumentImpl : Document {
    override val version: Long
        get() = 0
    override val size: Long
        get() = 0
    override fun load(stream: OutputStream) {
    }
    override fun save(input: InputStream) {
    }
    // No need to implement getDescription() as Java.
}

// inheritance
open class Shape(val name: String) {
    open fun area(): Double {
        return 0.0
    }
}
class Rectangle(name: String, val width: Double, val height: Double) : Shape(name) {
    override fun area(): Double {
        return width * height
    }
}

// abstract class
abstract class Shape(val name: String) {
    abstract fun area(): Double
}
class Rectangle(name: String, val width: Double, val height: Double) : Shape(name) {
    override fun area(): Double {
        return width * height
    }
}

// Visibility
// Public: This can be accessed from anywhere
// Internal: This can only be accessed from the module code
// Protected: This can only be accessed from the class defining it and any derived classes
// Private: This can only be accessed from the scope of the class defining it

// sealed Class
sealed class Result {
    data class Success(val message: String) : Result()
    data class Failure(val error: Throwable) : Result()
    fun handleResult(result: Result) {
        when (result) {
            is Success -> println("Success: ${result.message}")
            is Failure -> println("Failure: ${result.error}")
        }
    }
}

// data Class
data class Person(var name: String, var age: Int)

fun main() {
// Nested Class
    val tmpOuter = Outer()
    val tmpInner = tmpOuter.innerNestedClass() // Outer's inner nested class : 3

// this@label
    val tmpA = A()
    val tmpB = tmpA.B() // A.B.init() is called in here
    println("${tmpB.myVal}") // 2

// enum class
    val venus = Planet.valueOf("VENUS")
    println("VENUS : mass(${venus.mass}), radius(${venus.radius})") // VENUS : mass(4.869E24), radius(6051800.0)

// Singleton class = 'object' in Kotlin
    val myButtonInstance = myButton // Not myButton()
    val myButtonInstance2 = myButton // Not myButton()
    myButtonInstance.press() // 1
    myButtonInstance2.press() // 2
    myButton.press() // 3
    myButton.press() // 4

// data class
    var person = Person("Alice", 25)
    println(person.name) // Output: Alice
    println(person.age) // Output: 25
    val (name, age) = person
    println("Name: $name, Age: $age") // Output: Name: Alice, Age: 25
    person.name = "Alice2"
    person.age = 26
    println(person.name) // Output: Alice2
    println(person.age) // Output: 26
    println(person.hashCode()) // 750163080
    println(person.toString()) // Person(name=Alice2, age=26)

}



```