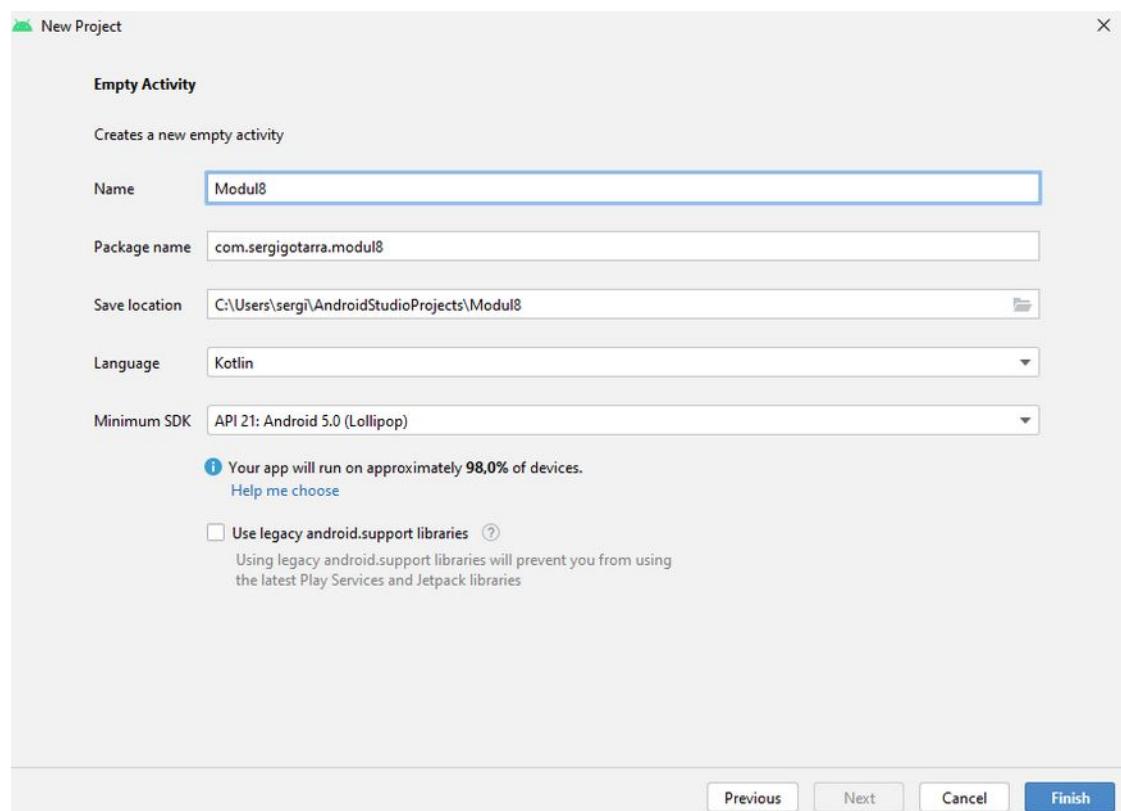
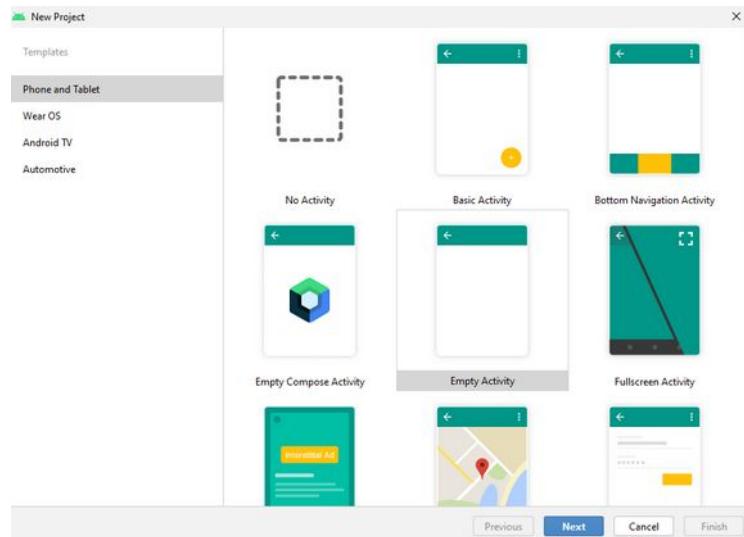


Tutorial alumnes introducció Android Studio i Kotlin

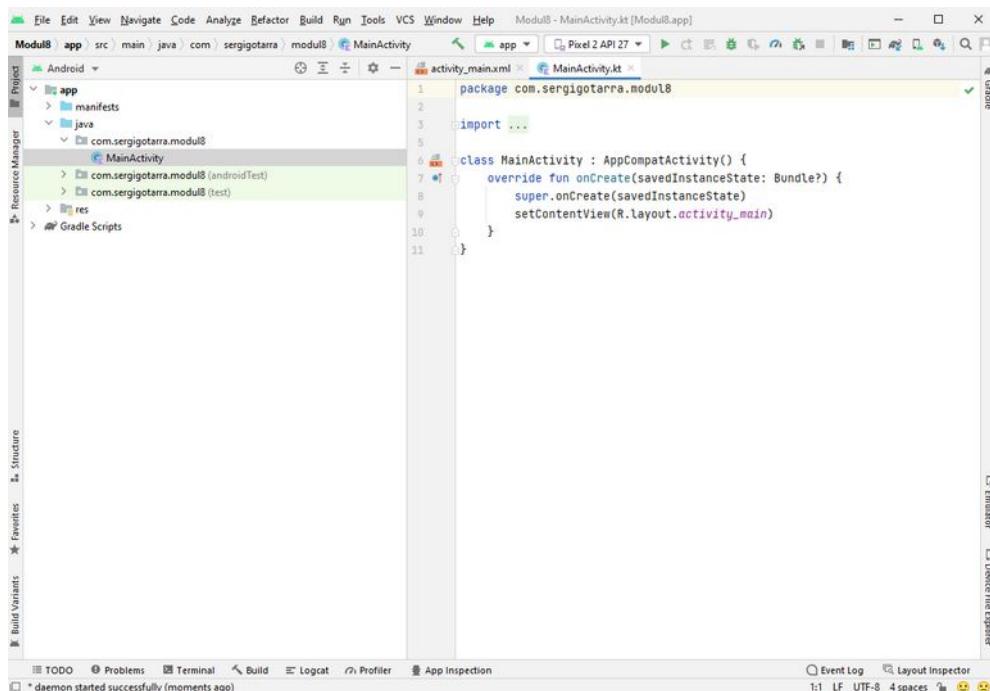
Sergi Gotarra Febrer 2022

Creem un projecte android studio



Name: 1a lletra en majúscula

Important: minimum SDK versió android mínima



Definició d'activity

Una Activity en Android es correspon amb una pantalla de la nostra App. En realitat és un punt d'entrada que Android pot carregar en qualsevol moment. Es compon de:

- Una classe, que normalment és herència de AppCompatActivity. És on definim el codi del que volem que faci l'App.
- Un layout, que identifica l'aparença de la vista, el disseny. Té format XML, però es pot utilitzar el dissenyador per a fer-lo més senzill
- Una definició del seu ús, que es realitza en el AndroidManifest.

Veiem la MainActivity que ha creat android studio per defecte:

Fitxer MainActivity

(Fitxer principal on farem override i crearem noves classes i funcions)

```

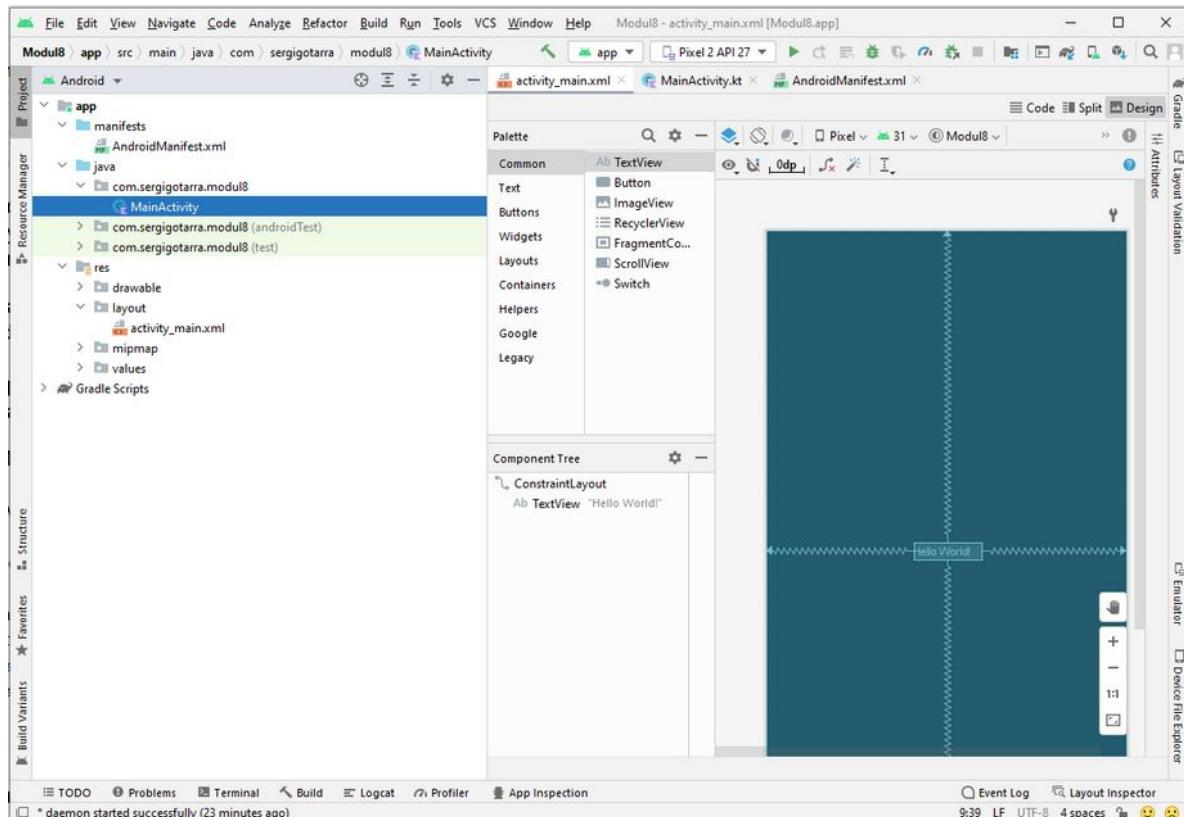
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}

```

Sobrescriu la funció onCreate del pare, crida a la funció del pare (una espècie de constructor) i defineix la finestra de contingut dins la classe R (R es una classe que es genera automàticament amb tots els recursos)

Si fem control + click a R.layout.activity_main anirem al fitxer **activity_main.xml** de Layout,

de dins de res (recursos)



Podem veure el codi (xml) o disseny, o split

L'altre arxiu necessari per a la Activity és el AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.sergigotarra.modul8">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Modul8">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

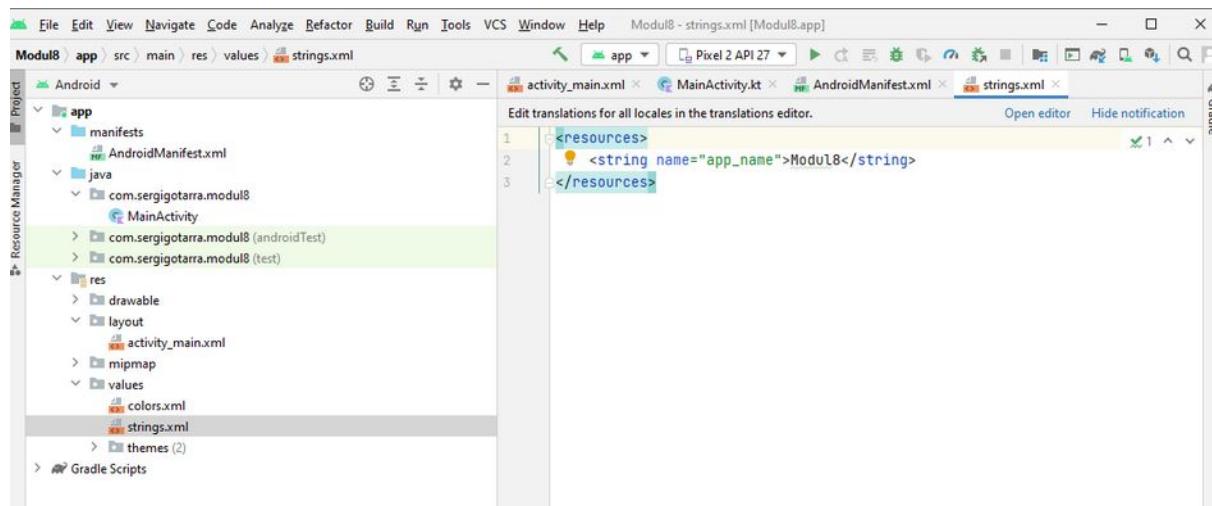
</manifest>
```

Dins de les etiquetes activity està la nostra main activity, si hi ha més activities també quedarán aquí referenciades.

Dins de intent-filter es defineix aquesta Activity com principal i que és la primera en intentar ser llançada quan arrenca el programa.

Més fitxers que fa servir la nostra aplicació:

Dins de values hi ha el arxiu **strings.xml**



Aquest fitxer emmagatzema totes les cadenes que es mostren en els widgets (controls, formes, botons, vistes, etc.) de les nostres activitats.

És important que tot el text estigui definit dins d'aquest arxiu. Per exemple, a strings afegim una etiqueta

```
<resources>
    <string name="app_name">Modul8</string>
    <string name="text_button">Pulsa aquí</string>
</resources>
```

Ara al arxiu Activity_main.xml canviem la referència a un text pla a una referència al arxiu de strings

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/text_button"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Altres fitxers i carpetes:

AndroidTest i Test són fitxers de proves

Res= recursos adicionals al codi

Drawable: els fitxers d'imatges decoratius, botons, icons (png, vectorials, svg...)

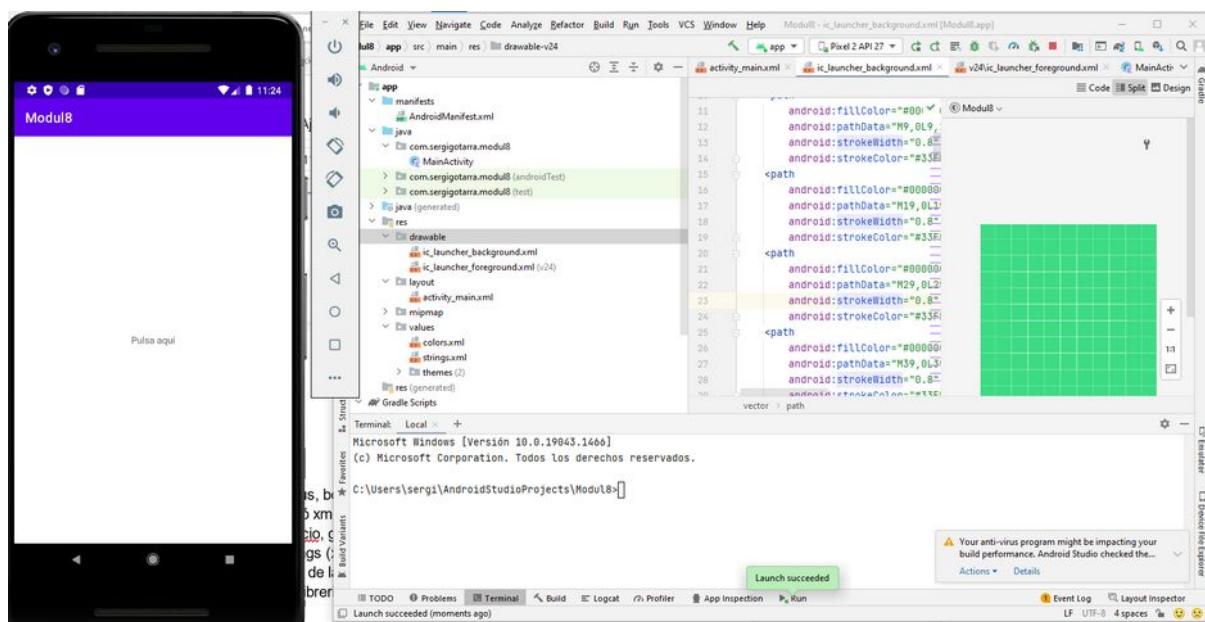
Layout: interfaces d'usuari de l'aplicació xml

Mipmap: Defineix les icones de l'aplicació, generats per a diferents densitats de píxels

Values: Variables diferents: colors, strings (xml) que ens permetrà fer traduccions a altres idiomes. Styles podem definir els estils de la plantilla de l'aplicació

Gradle: instal·lació final de l'aplicació, llibreries externes, etc. 2 Build.gradle (un del projecte i l'altre del mòdul)

Creem un emulador si no el tenim i fem córrer el programa



Ara ens fixem amb Logcat, ja que el farem servir per a veure la sintaxi bàsica de kotlin

Primerament modifiquem MainActivity

```
package com.sergigotarra.modul8

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //criudem a una funció que faci alguna cosa
        veurevariables()
    }

    private fun veurevariables()
```

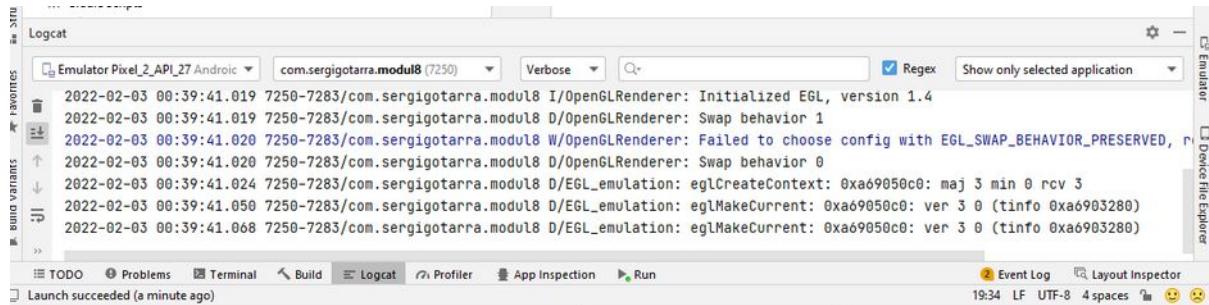
```

{
    var variable1 =" Pepe"
    Log.d("TAG::", variable1)
}

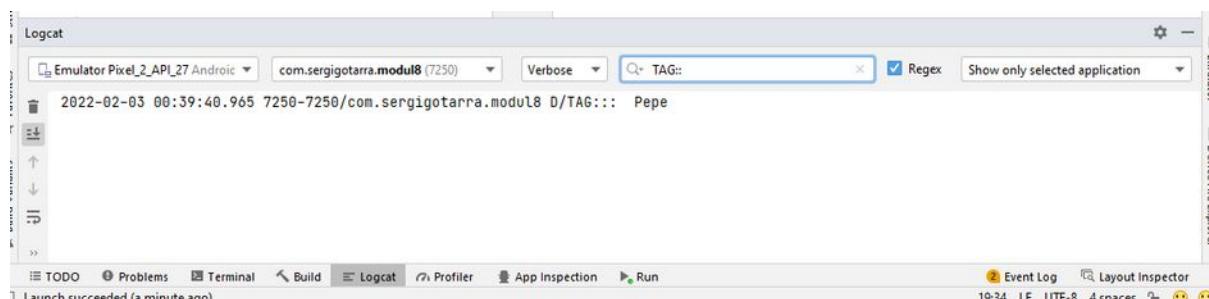
```

El "TAG::" servirà per a localitzar els nostres missatges dins la marabunta de missatges del terminal.

Provem-ho amb run



Hi ha molts missatges, ara els filtrarem amb el "TAG::"



I veiem el contingut de la variable1

Variables

Exemples de Variables en Kotlin (es poden definir de dues maneres, indicant el tipus o no)

```

var numero =1
var numero: Int =1
var numero: Long = 121212
var numero: Float = 1.9022f (obligatori la f)
var numero: Double = 1.2123231321

```

Alfanumèriques

```

var caracter: Char = 'p' (cometes simples!)
var cadena: String = "vaya vaya aqui no hay playa" (cometes dobles)

```

Booleanes

```
var iluminada: Boolean = false
```

```
var apagat: Boolean = true
```

Operacions aritmètiques: +, -, /, *, %

Podem veure un interrogant al final de un tipus, significa que el valor pot ser null

```
var iluminada: Boolean? = null
```

Casting

Per operacions entre variables de diferent tipus s'ha de fer casting a les variables

```
var a: Float = 10.4f
```

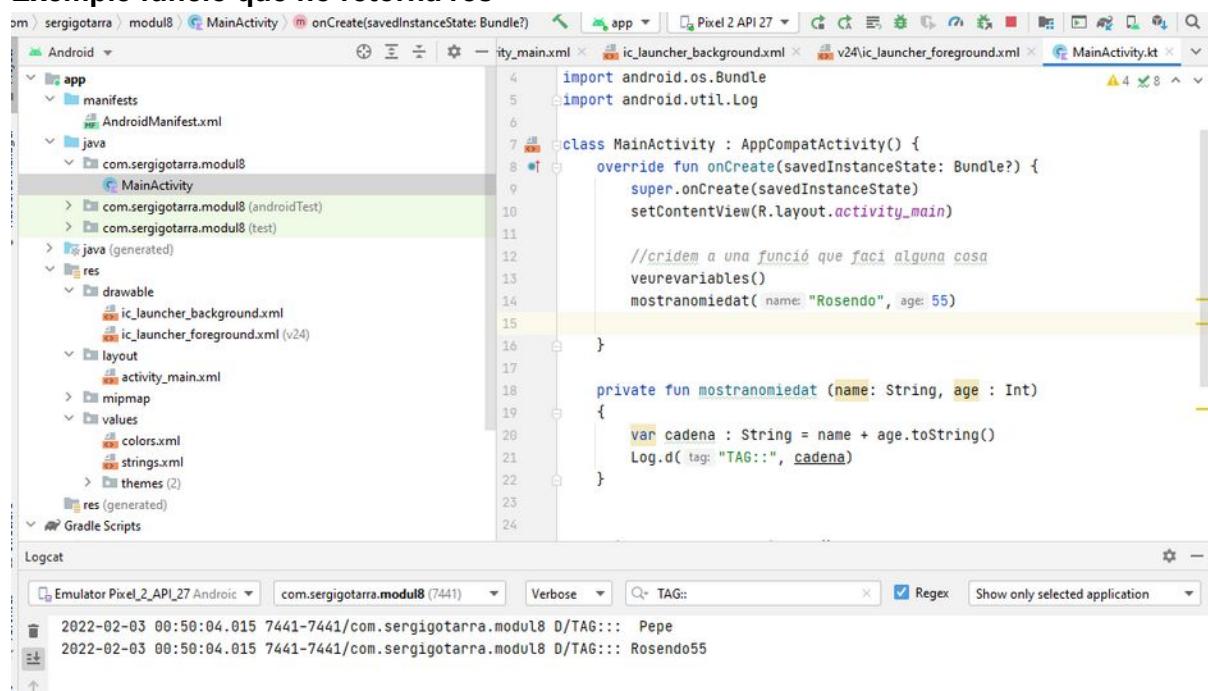
```
var b: Int = 12
```

```
var suma: Int
```

```
suma = a.toInt() + b
```

Declaració de funcions

Exemple funció que no retorna res



The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `MainActivity.kt` file:

```
import android.os.Bundle
import android.util.Log

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //crearem a una funció que faci alguna cosa
        veurevariables()
        mostrandomiedat("Rosendo", 55)
    }

    private fun mostrandomiedat(name: String, age: Int) {
        var cadena: String = name + age.toString()
        Log.d("TAG:::", cadena)
    }
}
```

The Logcat window at the bottom shows two log entries:

```
2022-02-03 00:50:04.015 7441-7441/com.sergigotarra.modul8 D/TAG::: Pepe
2022-02-03 00:50:04.015 7441-7441/com.sergigotarra.modul8 D/TAG::: Rosendo55
```

Below the code editor, there is another copy of the `MainActivity.kt` code:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //crearem a una funció que faci alguna cosa
        veurevariables()
        mostrandomiedat("Rosendo", 55)
    }

    private fun mostrandomiedat(name: String, age: Int) {
    }
}
```

```

var cadena : String = name + age.toString()
Log.d("TAG:::", cadena)
}

```

Exemple funció que torna un valor

The screenshot shows the Android Studio interface. On the left, the project structure is visible, including the app directory with manifests, Java files (MainActivity), and resources (res). The main area shows the code for MainActivity.kt:

```

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //crearem a una funció que faci alguna cosa
        veurevariables()
        mostranomedat( name: "Rosendo", age: 55 )
        Log.d( tag: "TAG:::", suma( a: 7, b: 9 ).toString() )

        private fun suma (a: Int, b:Int ): Int
        {
            return a+b
        }
    }
}

```

The Logcat window at the bottom shows the following output:

```

2022-02-03 00:53:23.455 7570-7570/com.sergigotarra.modul8 D/TAG::: Pepe
2022-02-03 00:53:23.456 7570-7570/com.sergigotarra.modul8 D/TAG::: Rosendo55
2022-02-03 00:53:23.456 7570-7570/com.sergigotarra.modul8 D/TAG::: 16

```

```
Log.d("TAG:::", suma(7,9).toString())
```

```

private fun suma (a: Int, b:Int ): Int
{
    return a+b
}

```

Condicionals

Primer creem una **constant** TAG amb val

```

.xml x ic_launcher_background.xml x v24\ic_launcher_foreground.xml x MainActivity
    package com.sergigotarra.modul8
    package com.sergigotarra.modul8

    import androidx.appcompat.app.AppCompatActivity
    import android.os.Bundle
    import android.util.Log

    val TAG = "TAG::: "

    class MainActivity : AppCompatActivity() {
        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)

            //crearem a una funció que faci alguna cosa
            veurevariables()
            mostranomedat( name: "Rosendo", age: 55)
            Log.d( tag: "TAG:::", suma( a: 7, b: 9).toString())
        }
    }

```

Afegeim el codi de exemple de condicionals

```

app
└── manifests
    └── AndroidManifest.xml
java
└── com.sergigotarra.modul8
    ├── MainActivity.kt
    └── com.sergigotarra.modul8 (androidTest)
        └── com.sergigotarra.modul8 (test)
res
└── drawable
    ├── ic_launcher_background.xml
    └── ic_launcher_foreground.xml (v24)
layout
└── activity_main.xml
mipmap
values
└── themes (2)
res (generated)
Gradle Scripts
└── build.gradle (Project: Modul8)
    └── build.gradle (Module: Modul8.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for Modul8.app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

```

```

setContentView(R.layout.activity_main)

//crearem a una funció que faci alguna cosa
veurevariables()
mostranomedat( name: "Rosendo", age: 55)
Log.d( tag: "TAG:::", suma( a: 7, b: 9).toString())
condicionalesIf()

private fun condicionalesIf() {
    val firstNumber = 10
    val secondNumber = 5
    val booleanValue = false

    if (firstNumber < secondNumber) {
        Log.d(TAG, msg: "Primera opción")
        if (secondNumber == 4) {
        } else {
        }
    } else if (booleanValue) {
        Log.d(TAG, msg: "Segunda opción")
    } else if (booleanValue) {
        Log.d(TAG, msg: "Segunda opción")
    } else if (booleanValue) {
        Log.d(TAG, msg: "Segunda opción")
    } else {
        Log.d(TAG, msg: "Tercera opción")
    }
}

```

Emulator Pixel_2_API_27 Androic

2022-02-03 02:05:09.438 8018-8018/com.sergigotarra.modul8 D/TAG::: Pepe
2022-02-03 02:05:09.438 8018-8018/com.sergigotarra.modul8 D/TAG::: Rosendo55
2022-02-03 02:05:09.438 8018-8018/com.sergigotarra.modul8 D/TAG::: 16
2022-02-03 02:05:09.438 8018-8018/com.sergigotarra.modul8 D/TAG::: Tercera opción
2022-02-03 02:05:09.438 8018-8018/com.sergigotarra.modul8 D/TAG::: 17

```

private fun condicionalesIf() {
    val firstNumber = 10
    val secondNumber = 5
    val booleanValue = false

    if (firstNumber < secondNumber) {

```

```

        Log.d(TAG, "Primera opcion")

        if (secondNumber == 4) {
        } else {
        }
    } else if (booleanValue) {
        Log.d(TAG, "Segunda opcion")
    } else if (booleanValue) {
        Log.d(TAG, "Segunda opcion")
    } else if (booleanValue) {
        Log.d(TAG, "Segunda opcion")
    } else {
        Log.d(TAG, "Tercera opcion")
    }

    val edad: Int = if (!booleanValue) {
        17
    } else {
        26
    }

    Log.d(TAG, edad.toString())
}

```

Condicionals When

Aquí el kotlin desenvolupa una estructura singular, diferent d'altres llenguatges

```

17 Log.d(tag: "TAG:::", suma(a: 7, b: 9).toString())
18 condicionalesIf()
19 condicionalsWhen()
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

```

```

private fun condicionalsWhen() {
    val senyoret = "Ric"

    when (senyoret) {
        "Pobre", "lleig" -> {
            Log.d(TAG, msg: "desgraciadet")
        }
        "Alt" -> Log.d(TAG, msg: "Jo vull ser alt")
        "Ric" -> Log.d(TAG, msg: "Jo vull ser ric")
        "Baixet" -> Log.d(TAG, msg: "Seleccionat baixet")
        else -> Log.d(TAG, msg: "Seleccionat un altre")
    }
}

```

Logcat output:

- 2022-02-03 02:13:26.597 8171-8171/com.sergigotarra.modul8 D/TAG::: Pepe
- 2022-02-03 02:13:26.597 8171-8171/com.sergigotarra.modul8 D/TAG::: Rosendo55
- 2022-02-03 02:13:26.597 8171-8171/com.sergigotarra.modul8 D/TAG::: 16
- 2022-02-03 02:13:26.597 8171-8171/com.sergigotarra.modul8 D/TAG::: Tercera opcion
- 2022-02-03 02:13:26.597 8171-8171/com.sergigotarra.modul8 D/TAG::: 17
- 2022-02-03 02:13:26.597 8171-8171/com.sergigotarra.modul8 D/TAG::: Jo vull ser ric

condicionalsWhen()

```

private fun condicionalsWhen() {
    val senyoret = "Ric"
}

```

```

when (senyoret) {
    "Pobre", "lleig" -> {
        Log.d(TAG, "desgraciadet")
    }
    "Alt" -> Log.d(TAG, "Jo vull ser alt")
    "Ric" -> Log.d(TAG, "Jo vull ser ric")
    "Baixet" -> Log.d(TAG, "Seleccionat baixet")
    else -> Log.d(TAG, "Seleccionat un altre")
}
}

```

Un altre exemple de com funciona el condicional When

```

private fun condicionalsWhen2() {
    val myNumber = 94
    when (myNumber) {
        in 0..10 -> {
            Log.d(TAG, "Se ha seleccionado Kotlin")
        }
        40 -> {
            Log.d(TAG, "Se ha seleccionado Java")
        }
        in 80..119 -> {
            Log.d(TAG, "Se ha seleccionado Python")
        }
        120 -> {
            Log.d(TAG, "Se ha seleccionado Ruby")
        }
        else -> Log.d(TAG, "Se ha seleccionado otro lenguaje")
    }
}

```

The screenshot shows the Android Studio interface. On the left is the Project Structure sidebar, which lists files like MainActivity.kt, build.gradle, and gradle-wrapper.properties. The main code editor displays a Kotlin function named condicionalsWhen2() that uses when clauses to log different messages based on the value of myNumber. Below the code editor is a Logcat window showing several log entries from an Emulator Pixel_2_API_27. The logs include messages like "Pepe", "Rosendo55", and "Jo vull ser ric".

```

private fun condicionalsWhen2() {
    val myNumber = 94
    when (myNumber) {
        in 0..10 -> {
            Log.d(TAG, msg: "Se ha seleccionado Kotlin")
        }
        40 -> {
            Log.d(TAG, msg: "Se ha seleccionado Java")
        }
        in 80..119 -> {
            Log.d(TAG, msg: "Se ha seleccionado Python")
        }
        120 -> {
            Log.d(TAG, msg: "Se ha seleccionado Ruby")
        }
        else -> Log.d(TAG, msg: "Se ha seleccionado otro lenguaje")
    }
}

private fun condicionalsWhen() {
    val senyoret = "Ric"
}

```

Llistes

Les llistes són dinàmiques, podem afegir elements, treure'n, etc

This screenshot shows the same Android Studio environment. The code editor now contains a function named llistes() that demonstrates various list operations. It creates a list of strings, adds an element, replaces an element, removes an element, and prints the list. The Logcat window shows the output of these operations, including the original list ["Rodrigo", "Raquel", "David", "Lorena"] and the modified list after changes were made.

```

//condicionalsWhen2()
llistes()

private fun llistes() {

    val myList = listOf("Rodrigo", "Raquel", "David", "Lorena",
    val myArrayList = arrayListOf("Rodrigo", "Raquel", "David",
    // listof i arraylistof son tipus similars amb algunes
    // diferencies en els mètodes

    val listItem = myList[2]
    Log.d(TAG, listItem)

    myList[2] = "Sandra"
    val arrayListItem = myArrayList[2]

    myArrayList.removeAt( index: 3)

    Log.d(TAG, myArrayList.toString())
}

```

```
private fun llistes() {
```

```

    val myList = listOf("Rodrigo", "Raquel", "David", "Lorena", "Allison")
    val myArrayList = arrayListOf("Rodrigo", "Raquel", "David", "Lorena",
"Allison")
    // listof i arraylistof son tipus similars amb algunes
    // diferencies en els mètodes

    val listItem = myList[2]
    Log.d(TAG, listItem)

    myArrayList[2] = "Sandra"
    val arrayListItem = myArrayList[2]

    myArrayList.removeAt(3)

    Log.d(TAG, myArrayList.toString())

}

```

Bucle FOR

Diverses maneres de fer un bucle for

```

bucleFor()

}

private fun bucleFor() {
    val myArrayList = arrayListOf("Rodrigo", "Raquel", "David", "Lorena", "Allison")

    // recorrem la llista per elements
    Log.d(TAG, "recorrer una list")
    for (persona in myArrayList) {
        Log.d(TAG, persona)
    }
    //típic bucle de 0 a 4
    Log.d(TAG, "bucle de 0 a 4")
    for (position in 0 until 5) {
        Log.d(TAG, position.toString())
    }

    //bucle de 0 a 10 saltant de 3 en 3
    Log.d(TAG, "bucle de 0 a 10 (de 3 en 3)")
    for (position in 0..10 step 3) {
        Log.d(TAG, position.toString())
    }

    //bucle invers de 10 a 3 de 2 en 2
    Log.d(TAG, "bucle invers")
    for (position in 10 downTo 3 step 2) {
        Log.d(TAG, position.toString())
    }
}

```

Bucle While

```
bucleWhile()
```

```

private fun bucleWhile() {
    var myNumber = 0

    while (myNumber <= 10) {
        Log.d(TAG, myNumber.toString())
        myNumber += 3
    }
}

//llistes()
//bucleFor()
bucleWhile()

private fun bucleWhile() {
    var myNumber = 0

    while (myNumber <= 10) {
        Log.d(TAG, myNumber.toString())
        myNumber += 3
    }
}

```

The screenshot shows the Android Studio interface. On the left, the Project structure is visible, showing files like activity_main.xml, values/colors.xml, and build.gradle. The main code editor displays a Java file with a while loop. Below it, the Logcat tab shows the output of the Log.d statements: four entries at level 0, three at level 3, six at level 6, and nine at level 9, corresponding to each iteration of the loop.

Bucle Do While

Igual que el While però fa la comprovació al final

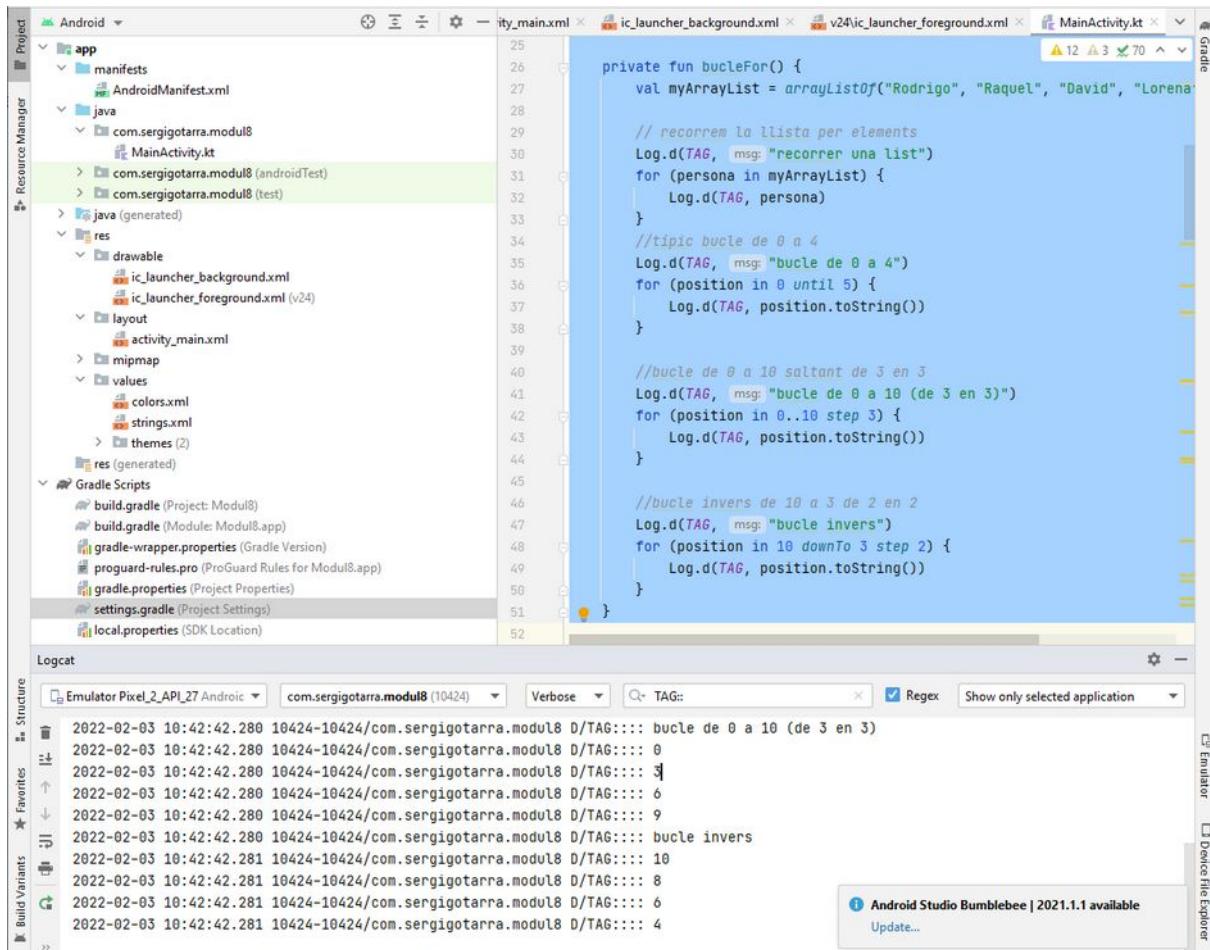
bucleDoWhile()

```

private fun bucleDoWhile() {
    var myNumber = 1

    do {
        Log.d(TAG, myNumber.toString())
        myNumber++
    } while (myNumber <= 10)
}

```



TRY, control d'errors

Seqüències Try per capturar errors

```
try {
    // some code
} catch (e: SomeException) {
    // handler
} finally {
    // optional finally block
}

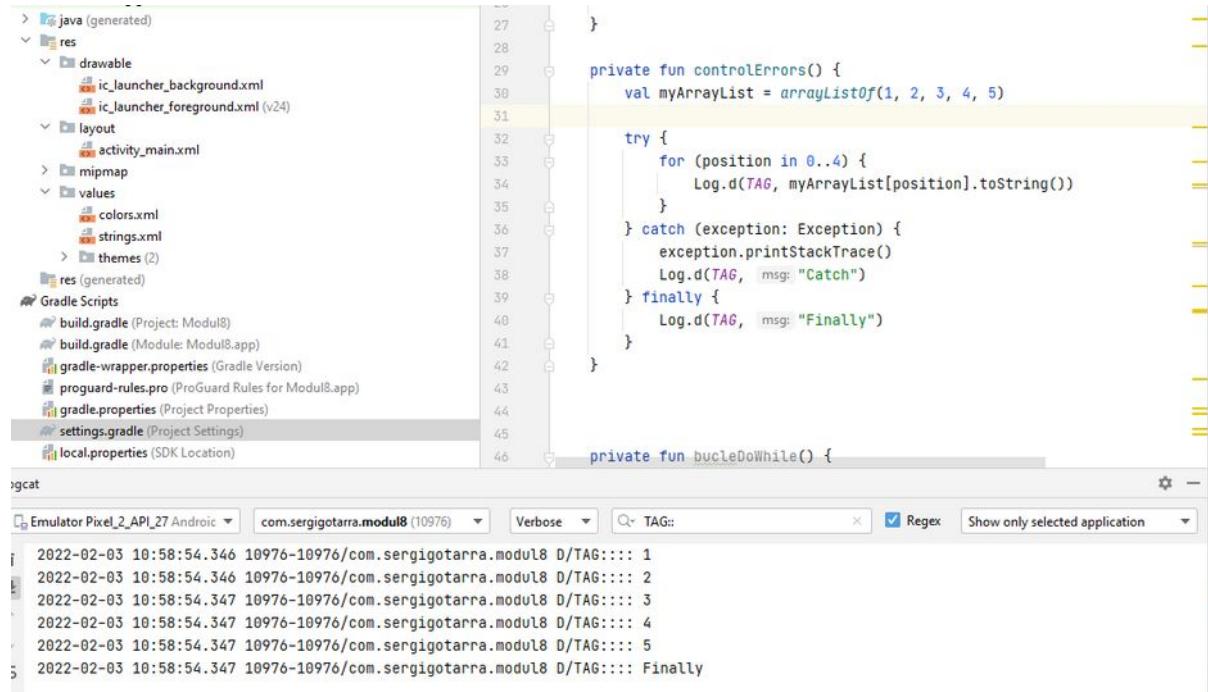
controlErrors()

private fun controlErrors() {
    val myArrayList = arrayListOf(1, 2, 3, 4, 5)
    try {
        for (position in 0..4) {
            Log.d(TAG, myArrayList[position].toString())
        }
    } catch (exception: Exception) {
        exception.printStackTrace()
        Log.d(TAG, "Catch")
    } finally {
    }
}
```

```

        Log.d(TAG, "Finally")
    }
}

```



Provoquem un error:

```

private fun controlErrors() {  

    val myList = arrayListOf(1, 2, 3, 4, 5)  

    try {  

        for (position in 0..5) {  

            Log.d(TAG, myList[position].toString())  

        }
    } catch (exception: Exception) {  

        exception.printStackTrace()  

        Log.d(TAG, "Catch")  

    } finally {  

        Log.d(TAG, "Finally")  

    }
}

```

```

> java (generated)
  res
    drawable
      ic_launcher_background.xml
      ic_launcher_foreground.xml (v24)
    layout
      activity_main.xml
    mipmap
    values
      colors.xml
      strings.xml
  themes (2)
res (generated)
  Gradle Scripts
    build.gradle (Project: Modul8)
    build.gradle (Module: Modul8.app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for Modul8.app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)

private fun controlErrors() {
    val myList = arrayListOf(1, 2, 3, 4, 5)

    try {
        for (position in 0..5) {
            Log.d(TAG, myList[position].toString())
        }
    } catch (exception: Exception) {
        exception.printStackTrace()
        Log.d(TAG, msg: "Catch")
    } finally {
        Log.d(TAG, msg: "Finally")
    }
}

private fun bucleDoWhile() {
}

```

ogcat

Emulator Pixel_2_API_27 Androic com.sergigotarra.modul8 (11114) Verbose TAG: Regex Show only selected application

```

2022-02-03 10:59:59.356 11114-11114/com.sergigotarra.modul8 D/TAG::: 1
2022-02-03 10:59:59.356 11114-11114/com.sergigotarra.modul8 D/TAG::: 2
2022-02-03 10:59:59.356 11114-11114/com.sergigotarra.modul8 D/TAG::: 3
2022-02-03 10:59:59.356 11114-11114/com.sergigotarra.modul8 D/TAG::: 4
2022-02-03 10:59:59.356 11114-11114/com.sergigotarra.modul8 D/TAG::: 5
2022-02-03 10:59:59.358 11114-11114/com.sergigotarra.modul8 D/TAG::: Catch
2022-02-03 10:59:59.358 11114-11114/com.sergigotarra.modul8 D/TAG::: Finally

```

Classes

Crear una classe:

```

var llibre = Book("el quixot", 123456677898)
Log.d(TAG, llibre.title)

```

```

class Book {
    var title: String
    var isbn: Long

    constructor(title: String, isbn: Long) {
        this.title = title
        this.isbn = isbn
    }
}

```

Aquest constructor s'anomena **constructor secundari**

Els getters i setters per a aquestes propietats són autogenerats pel compilador Kotlin. No especifiquem cap modificador d'accés per a aquestes propietats, així que per defecte, són públiques. En altres paraules, poden ser accedides des de qualsevol lloc.

El constructor es pot incloure a la definició de la classe així (anomenat **constructor primari**):

```

class Book2 constructor(title: String, isbn: Long) {

```

```

var title: String
var isbn: Long

init {
    this.title = title
    this.isbn = isbn
}
}

```

init corre cada vegada que d'instància la classe

Una tercera manera de crear aquesta classe, (si no hi ha modificadors: private, public o protected)

```
class Book3 (var title: String = "default value", var isbn: Long)
```

Valors val i var dins de les classes

```

val book = Book5("A Song of Ice and Fire", 9780007477159)
book.isbn = 1234 // error: read-only property, és un VAL
book.title = "Things Fall Apart" // reassigned title with value

```

```

class Book5 (
    var title: String,
    val isbn: Long
)

```

Utilitzar diversos constructors per a la mateixa classe (polimorfisme)

```

class Car(val name: String, val plateNo: String) {
    var new: Boolean? = null
    var colour: String = ""

    constructor(name: String, plateNo: String, new: Boolean) :
this(name, plateNo) {
        this.new = new
    }

    constructor(name: String, plateNo: String, new: Boolean, colour:
String ): this(name, plateNo, new) {
        this.colour = colour
    }
}

```

Podem instanciar aquesta classe així:

```
// directly calls primary constructor
```

```

val car1 = Car("Peugeot 504", "XYZ234")
    // directly calls 1st sec. constructor
val car2 = Car("Peugeot 504", "XYZ234", false)
    // directly calls last sec. constructor
val car3 = Car("Peugeot 504", "XYZ234", false, "grey")

```

Modificadors de visibilitat

Els modificadors de visibilitat en Kotlin són molt semblants als de Java, amb la diferència que tot en Kotlin per defecte és públic

public: és visible des de tots costats.

protected: és visible solo per a classes "filles" d'una classe.

private: és visible solo dins de la mateixa classe.

internal: és visible solo dins del mateix mòdul.

Modificadors d'accés en Kotlin

final: És usat per defecte en les classes i no pot ser sobre escrit.

open: Pot ser sobre escrit i ha de ser explícitament escrit.

abstract: Ha de ser sobre escrit, aquest únicament ha de ser usat en classes abstractes.

Herència i Override

No es pot fer herència múltiple de classes, però si d'interfícies.

Interfície: Classe que té tots els mètodes públics i poden ser sobreescrits

Una classe de la que es pot heretar s'ha de marcar amb el modificador **open**

```
open class persona (val nom: String)
```

Una classe que hereta d'una altra ho fa indicant : i cridant al constructor de la classe pare

```
class magobueno (val nombremago: String) : persona(nombremago)
Per sobreescrivir funcions s'utilitza override
```

```

val jaina = Mage("Jaina")
jaina.die()

open class Character(val name: String) {
    open fun die() = Log.d(TAG, "MORIR")
}

class Mage(name: String) : Character(name) {
    override fun die() = Log.d(TAG, "mago muere")
}

```

Per aprendre més veure:

<https://www.develou.com/herencia-en-kotlin/>

Objectes en Kotlin (NO SÓN INSTÀNCIES DE CLASSES)

Els objectes són molt similars a les classes. Aquestes són algunes de les característiques dels objectes en Kotlin:

- Poden tenir propietats, mètodes i un bloc init.
- Aquestes propietats o mètodes poden tenir modificadors de visibilitat.
- No poden tenir constructors (primaris o secundaris).
- Poden estendre altres classes o implementar una interfície.

```

val suma = Calculadora.sumarDosNumeros(1,3)
Calculadora.possamemoria(suma)
Log.d(TAG, Calculadora.getmemoria().toString())

```

```

object Calculadora {
    var memoria: Int =18;
    fun sumarDosNumeros(sum1: Int, sum2: Int): Int{
        return sum1+sum2
    }
    fun possamemoria(mem: Int)
    {
        memoria=mem
    }
    fun getmemoria(): Int
    {
        return memoria
    }
}

```

The screenshot shows the Android Studio interface. On the left, the Project Structure sidebar displays the following files:

- com.sergigotarra.modul8 (test)
- java (generated)
- res
 - drawable
 - ic_launcher_background.xml
 - ic_launcher_foreground.xml (v24)
 - layout
 - activity_main.xml
 - mipmap
 - values
 - colors.xml
 - strings.xml
 - themes (2)
- res (generated)
- Gradle Scripts
 - build.gradle (Project: Modul8)
 - build.gradle (Module: Modul8.app)
 - gradle-wrapper.properties (Gradle Version)
 - proguard-rules.pro (ProGuard Rules for Modul8.app)
 - gradle.properties (Project Properties)
 - settings.gradle (Project Settings)
 - local.properties (SDK Location)

The code editor on the right contains Java code for a class named Calculadora. The code includes a main method that calls a sumarDosNumeros function and a possamemoria function, then logs the result. It also defines an object Calculadora with methods for summing numbers, setting memory, and getting memory.

```
> com.sergigotarra.modul8 (test)
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45

//bucleDoWhile()
//controlErrors()
val suma = Calculadora.sumarDosNumeros( sum1: 1, sum2: 3 )
Calculadora.possamemoria(suma)
Log.d(TAG, Calculadora.getmemoria().toString())

}

object Calculadora {
    var memoria: Int =18;
    fun sumarDosNumeros(sum1: Int, sum2: Int): Int{
        return sum1+sum2
    }
    fun possamemoria(mem: Int)
    {
        memoria=mem
    }
    fun getmemoria(): Int
    {
        return memoria
    }
}
```