# B.3 – Debugger Intro

## A cosa serve il Debugger

Il Debugger è uno strumento che ci permette di controllare la nostra app a *runtime*, vedendo in tempo reale come si aggiornano valori e oggetti, e mettendo in pausa l’esecuzione di classi e funzioni.

## Breakpoints

Scriviamo il seguente codice dentro *onCreate*

data class ColorHolder(val colors: List<String>) {  
 fun hasThreeColors() = colors.size == 3  
}  
  
var holder: ColorHolder  
  
val myColors = *mutableListOf*("Green", "Red", "Cyan")  
 .*also* **{** holder = ColorHolder(**it**) **}**myColors.add("Yellow")  
  
val message =  
 if (holder.hasThreeColors())   
 "Holder has 3 colors!"  
 else   
 "Holder is bad =("  
  
*println*(message)

Lanciando l’app, il nostro holder non è quello che ci aspettiamo, avendo piu’ di 3 colori.   
Cliccando nella barra laterale dove sono numerate le linee di codice è possibile aggiungere un *Breakpoint*. Aggiungilo alla riga *myColors.add(“Yellow”)*:

A picture containing text

Description automatically generated

Lanciamo ora l’app sull’emulatore con la funzionalità *Debug*

*Graphical user interface, text, application

Description automatically generated*

Giunto all’esecuzione della riga con il breakpoint, il programma si metterà in pausa **prima di eseguire la riga.** L’IDE ci mostrerà una diversa interfaccia:

Text

Description automatically generated

* In basso a sinistra, vediamo una lista di processi. In alto è indicato il thread dove siamo fermi; Il processo selezionato ci dice il punto del codice dove siamo fermi; mentre sotto sono elencate in sequenza le invocazioni, spesso interne del sistema Android, che alla fine hanno portato alla classe o la funzione dove ci siamo fermati.

Text

Description automatically generated

* In basso a destra c’è una lista di variabili. E’ possibile controllare come sono valorizzate al momento del nostro breakpoint:  
  Text

  Description automatically generated
* Nel codice stesso vengono aggiunti dei commenti per mostrarci l’attuale valorizzazione delle variabili, oltre ad avere la riga selezionata per evidenziare il punto di stop.  
  Text

  Description automatically generated with medium confidence

Per ora il nostro ColorHolder è corretto: ha ancora 3 colori. Aggiungiamo altri 2 breakpoint:

Text

Description automatically generated

I breakpoints seguono l’esecuzione del programma: quest’ultimo procederà nell’ IF se la condizione è vera, e nell’ELSE se è falsa.

Cliccando sul comando *Resume Program (F9)* possiamo farlo ripartire:

Graphical user interface, text, application

Description automatically generated

Ora ecco dove si fermerà:

A picture containing graphical user interface

Description automatically generated

La condizione è falsa, e leggendo il suggerimento è facilmente intuibile il perché.

## Evaluate

Il Debugger ci fornisce anche la possibilità di sfruttare il programma per scrivere codice mentre sta venendo eseguito.

Partendo da questi breakpoints, lanciamo l’app in debug:

A picture containing text

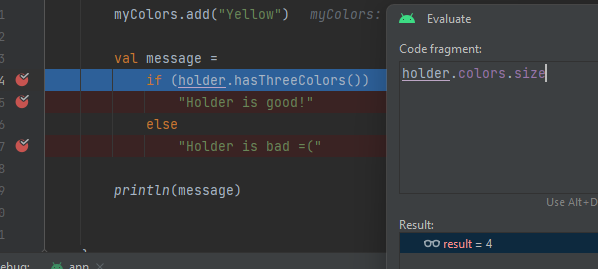
Description automatically generated

Giunti al primo breakpoint, clicca su *Evaluate Expression*

A screenshot of a computer

Description automatically generated

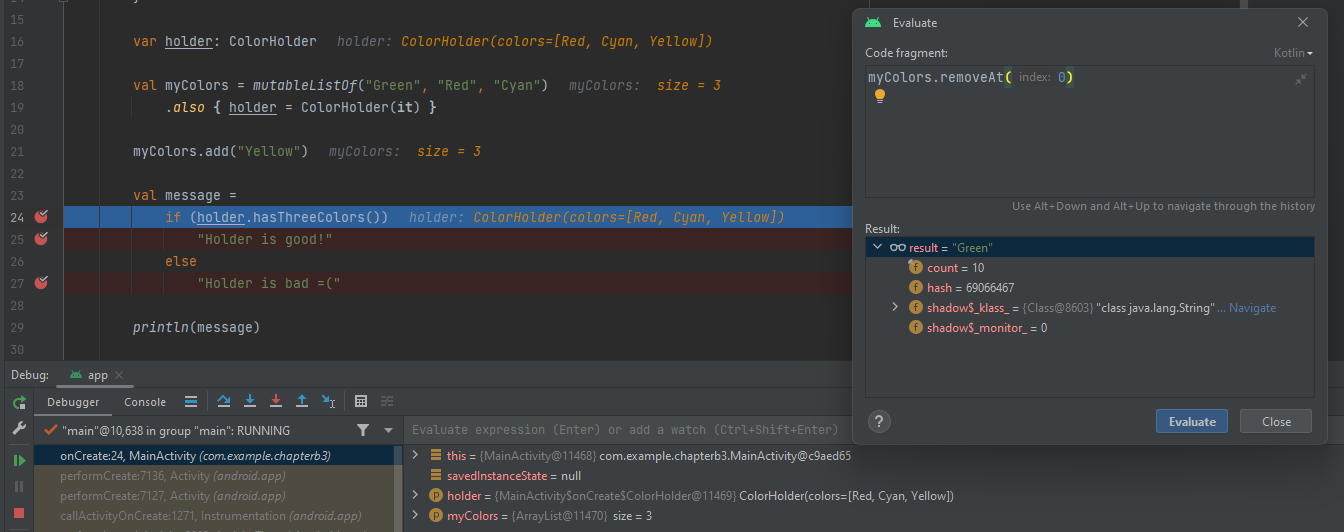
Dentro a questo tool possiamo scrivere del codice per comprendere meglio lo stato dell’app in questo istante. Ad esempio, richiediamo il *size* della lista *holder.colors* prima che venga lanciato IF ELSE



Prevediamo quindi che sarà lanciata la condizione ELSE.

Abbiamo pero’ la possibilità di modificare l’assegnazione delle variabili in tempo reale, modificando cosi’ il flusso dell’app.

Rimuoviamo il primo colore da myColors. L’IDE si aggiorna per mostrarci i valori che sono stati modificati in corsa.



Procedendo al breakpoint successivo con *Resume Program* vediamo che ora la condizione è vera ed il programma passa dal flusso IF invece che da ELSE.

Graphical user interface, text

Description automatically generated