React Native e Swift

DISCLAIMER:

Questi appunti fanno cagare, ma meglio questo che niente.

React Native

Mappa i componenti JS con i componenti nativi, così si usa React per scrivere l'interfaccia grafica.

Basato sull'idea di componente: un oggetto con uno stato e delle proprietà.

A differenza di altri framework non si basa su una web-view (che introduce latenza), ma chiede direttamente al sistema operativo di disegnare. Si bas aquindi su un metodo chiamato Virtual-DOM (document object model), che genera un albero dei contenuti da renderizzare. Quando aggiungo un componente lo aggiunge al dom e lo renderizza (non rirenderizza tutto).

Quando si compila il codice vengono creati 2 pacchetti, uno android uno IOS (come flutter). Il codice è scritto in JS + CSS

PRO : comunità molto attiva, molti componenti open source

Swift

Dichiarazione di tipo non obbligata, come in kotlin il tipo optional si indica con Type?

Java soffre di memory leak (posso ad esmpio modificare variabili private di una classe se ne ho la reference), un po' anche Kotlin (SI è provato a risolvere con le Data Classes, classi apposta). In swift si introduce un costrutto chiamato Struttura, che possono implementare dei "protocolli" (interfacce in java) per fornire alcune funzionalità, adatte a mantenere al loro interno dei dati. Risolvono il problema dei memory leak perché vengono sempre passate per valore, mai per riferimento. var newStruct = struct; copia la struttura struct, non passa il riferimento.

Stringhe, array, set e dizionari sono implementati tramite strutture

Automatic Reference Counting (ARC)

Java e kotlin usano il Garbage collector: si parte dallo stack, si guardano i riferimenti all'heap. Tutto quello che si può raggiungere dai riferimenti sullo stack si tiene, il resto non serve più e lo si sovrascrive.
Ha dei problemi: 1 è lento (deve leggere tutto lo stack) 2 è facile creare memory leak, oggetti che non servono ma che rimangono nello heap. Potrebbero essere deallocato ma il GC non lo sa.

Swift funziona in maniera diversa, considerando che le app sono "gerarchiche" se il "contenitore" padre sparisce posso sicuramente deallocare tutti i contenitori figli.
Alcune referenze possono essere indicate come weak. Una referenza "weak" indica un oggetto che quando non più in uso può essere deallocato, automaticamente settando a null tutte le variabili che si riferivano ad esso. Di default le reference sono strong. Come capisco se un oggetto che voglio deallocare è puntato solo da variabili weak? All'interno di ogni oggetto ho un contatore, l'ARC, che

mantiene il conto di quanti riferimenti strong ci sono all'oggetto ed ho un riferimento a tutte le variabili weak che puntano all'oggetto. Quando ARC = 0 l'oggetto viene deallocato e tutte le variabili weak che lo puntano sono poste a null.

Per riagganciarci al discorso gerarchico: da padre a figlio è strong reference, viceversa è weak reference.

Grand Central Dispatch (GCD)

Gestione della concorrenza, code di default che automagicamente gestiscono al meglio i thread assegnati alle code.
Identico alle Corutines in Android.

Inizializzazione

In java molti errori sono dovuti ad una scorretta inizializzazione, il codice del costruttore potrebbe accedere a campi non inizializzati, cio è dovuto all'ereditarietà dei costruttori.
Kotlin "risolve" separando l'inizializzazione dalla "costruzione", il costruttore non contiene alcun codice, l'inizializzatore (keyword init) invece sì.
Swift in fase di inizializzazione controlla che sia impostato un valore iniziale ad ogni proprietà stored (se così non è o si arrabbia o imposta un valore di default se il tipo è "di base", int viene settato a 0 di default) Se non meglio specificato gli Optional sono nil.
Oltretutto in init il programmatore può scrivere altro codice in modo che l'istanza sia pronta da usare.

Un inizializzatore può richiamarne un altro, ma quando un programmatore scrive un inizializzatore non può riferirsi a quello di default (a differenza di java), quello di default sparisce.
Ci sono 2 tipi di inizializzatori di classe, designati e di convenienza. Di default una classe non eredita l'inizializzatore della super classe. Un inizializzatore designato inizializza tutte le proprietà della classe (anche le ereditate, posso modificarle ma le DEVO inizializzare prima richiamando l'inizializzatore della superclasse).
Inceve i convinience sono inizializzatori "patternizzati" che richiamano i designati.

Il compilatore effettua 3 controlli:
1 Un inizializzatore designato deve richiamare un inizializzatore della sua superclasse immediata.

2 Un inizializzatore convenience deve chiamare un'altro inizializzatore della stessa classe

3 Alla fine di una catena di chiamate ad inizializzatori convinience ci deve essere un designato

Tutto ciò serve a garantire che non si provi mai ad accedere a delle variabili che non sono state inizializzate

come in java la keyword required prima di un metodo (in questo caso un inizializzatore) indica che il metodo deve essere implementato nelle sottoclassi. A differenza di Java per fare override non s usa @overrride. Si scrive di nuovo required prima dell'inizializzatore che si sta "riscrivendo".

Ogni classe ha in automatico un deinizializzatore (keyword deinit) di cui il programmatore può fare override.Deinit viene chiamato in fase di deallocazione