Il flusso

La Natura del Flusso di Esecuzione
Il **flusso di esecuzione** è l'ordine in cui le istruzioni del programma vengono eseguite. Normalmente, il flusso è **sequenziale**: le istruzioni vengono eseguite una dopo l'altra, dall'alto verso il basso, nell'ordine in cui sono scritte.
Il controllo di flusso è ciò che ti permette di **deviare** da questa sequenza lineare. Ti permette di saltare istruzioni, ripeterle, o scegliere tra percorsi alternativi basandosi su condizioni.
Senza controllo di flusso, i programmi sarebbero rigidi e limitati: potrebbero fare solo sequenze fisse di operazioni, sempre nello stesso ordine. Il controllo di flusso dà ai programmi la capacità di **prendere decisioni** e **adattarsi** a situazioni diverse.

La Selezione: Scegliere tra Percorsi
La **selezione** è il meccanismo che permette al programma di eseguire codice diverso basandosi su condizioni. "Se questa condizione è vera, fai questo; altrimenti, fai quello".**Il concetto di branch**
Ogni punto di selezione crea un **branch** (ramificazione) nel flusso di esecuzione. Il programma può prendere percorsi diversi attraverso il codice. Non tutte le istruzioni verranno necessariamente eseguite: dipende dalle condizioni.
Questa è una differenza fondamentale rispetto al flusso sequenziale: non puoi più essere sicuro che una certa istruzione verrà eseguita solo perché appare nel codice. Devi considerare **sotto quali condizioni** viene raggiunta.
L'Istruzione If: La Decisione Binaria
L'**if** è la forma più elementare di controllo decisionale. Valuta una condizione booleana: se è vera, esegue un blocco di codice; se è falsa, lo salta.**La condizione come cancello**
La condizione dell'if agisce come un **cancello**: il codice dentro l'if può essere eseguito solo se il cancello è aperto (condizione vera). Se il cancello è chiuso (condizione falsa), quel codice viene completamente ignorato come se non esistesse.**If senza else: opzionalità**
Un if senza else rappresenta un'operazione **opzionale**: potrebbe succedere o potrebbe non succedere, ma se non succede, il programma continua normalmente. Non c'è un percorso alternativo: c'è solo "fai questo se possibile" oppure "prosegui oltre".

L'Istruzione Else: Il Percorso Alternativo

L'**else** fornisce un percorso alternativo esplicito. "Se la condizione è vera, fai questo; altrimenti, fai quest'altro". È una scelta **dicotomica**: esattamente uno dei due blocchi verrà eseguito, mai entrambi, mai nessuno.**La garanzia dell'else**

Con if-else, hai la garanzia che qualcosa verrà fatto. Non c'è possibilità che entrambi i rami vengano saltati. Questo è utile quando devi assicurarti che una certa azione venga presa, indipendentemente dalla condizione.**Else come caso residuo**

L'else cattura "tutto il resto": qualsiasi caso che non soddisfa la condizione dell'if finisce nell'else. Non devi specificare una condizione per l'else; è implicitamente "non-if".

Else-If: Decisioni Multiple Sequenziali

La catena **else-if** permette di testare multiple condizioni in sequenza. "Se condizione1, fai A; altrimenti se condizione2, fai B; altrimenti se condizione3, fai C; altrimenti fai D".**Valutazione a corto circuito**

Le condizioni vengono valutate **in ordine**, e appena una risulta vera, il blocco corrispondente viene eseguito e tutto il resto della catena viene saltato. Non si continua a controllare le altre condizioni.

Questo significa che l'**ordine è importante**. Se metti prima condizioni più generiche e poi più specifiche, quelle specifiche potrebbero non essere mai raggiunte.**L'else finale come default**

L'else finale in una catena else-if rappresenta il **caso di default**: cosa fare quando nessuna delle condizioni precedenti è soddisfatta. È un catch-all, una rete di sicurezza.

If Annidati: Decisioni Gerarchiche

Puoi mettere un if **dentro** un altro if. Questo crea una struttura gerarchica di decisioni: "Se condizione1 è vera, allora controlla anche condizione2".**Livelli di profondità**

Ogni livello di annidamento aggiunge un livello di condizionalità. Per raggiungere il codice più interno, devono essere vere tutte le condizioni dei livelli superiori.**Il rischio della complessità**

If troppo annidati diventano difficili da leggere e comprendere. Ogni livello di indentazione aggiunge carico cognitivo. Se ti trovi con 4-5 livelli di annidamento, probabilmente c'è un modo migliore di strutturare la logica.

Switch: Decisione Multivia Basata su Valore

Lo **switch** è un'alternativa all'else-if quando devi scegliere tra molte opzioni

basandoti sul valore di una singola espressione.**Confronto diretto di valori**
Lo switch confronta un valore con una serie di **costanti**. Non può valutare espressioni booleane complesse come l'if; può solo verificare l'uguaglianza con valori specifici.**Case come etichette**
Ogni **case** è un'etichetta che dice "se il valore è questo, inizia l'esecuzione da qui". Non è un blocco isolato come l'if: l'esecuzione può "cadere attraverso" (fall-through) al case successivo se non metti un break.**Il fall-through: caratteristica o insidia**
Il fall-through è una caratteristica deliberata dello switch: se ometti il break, l'esecuzione continua nel case successivo. Questo può essere utile per casi che condividono lo stesso codice, ma è anche una fonte comune di bug quando lo dimentichi accidentalmente.**Default: il caso catch-all**
Il **default** nello switch è come l'else finale: viene eseguito se nessun case corrisponde. È opzionale, ma spesso è buona pratica includerlo per gestire valori inaspettati.

Break nel Controllo di Flusso

Il **break** è un'istruzione che interrompe il flusso normale, causando un'uscita immediata da un ciclo o da uno switch.**Salto incondizionato**
Quando esegui un break, non importa dove sei nel ciclo o quante iterazioni mancano: esci immediatamente. È un salto diretto alla prima istruzione dopo la struttura che stai interrompendo.**Break e cicli annidati**
In cicli annidati, il break esce solo dal ciclo più interno. Se vuoi uscire da più livelli, devi usare break multipli o usare flag booleani che controlli nei cicli esterni.

Continue: Salto all'Iterazione Successiva

Il **continue** salta il resto del corpo del ciclo corrente, ma non esce dal ciclo: passa direttamente alla prossima iterazione.**La differenza sottile**
Continue dice "non fare il resto di questa iterazione, ma continua il ciclo". Break dice "non fare più nessuna iterazione, esci completamente". Continue è un salto parziale; break è un salto totale.**Evitare indentazioni profonde**
Continue è utile per evitare if annidati. Invece di scrivere "if (condizione) { tutto il resto del codice }", puoi scrivere "if (!condizione) continue; resto del codice al primo livello".

Return: Uscita dalla Funzione

Il **return** è la forma più drastica di controllo di flusso all'interno di un metodo: termina immediatamente l'esecuzione del metodo e restituisce il controllo al chiamante.**Return anticipato**

Puoi usare return in qualsiasi punto del metodo, non solo alla fine. Questo è chiamato **early return** e serve per uscire dal metodo non appena hai la risposta o quando ulteriore elaborazione è inutile.

Short-Circuit Evaluation: Ottimizzazione Logica

Gli operatori logici **&&** (AND) e **||** (OR) usano la valutazione **short-circuit**: se possono determinare il risultato dall'operando sinistro, non valutano quello destro.**AND short-circuit**

Con `A && B`, se A è falso, l'intero risultato è falso indipendentemente da B. Quindi B non viene valutato. Questo è utile per verifiche di sicurezza: `oggetto != null && oggetto.metodo()` non chiama il metodo se l'oggetto è null.**OR short-circuit**

Con `A || B`, se A è vero, l'intero risultato è vero indipendentemente da B. Quindi B non viene valutato.**Implicazioni pratiche**

Questo influenza l'ordine in cui scrivi le condizioni. Metti prima le verifiche più economiche o quelle che hanno maggiore probabilità di determinare il risultato. E sfrutta lo short-circuit per evitare operazioni costose o pericolose.

L'Operatore Ternario: If Compatto

L'operatore ternario `condizione ? valoreSeVero : valoreSeFalso` è un if-else compatto che produce un valore.**Espressione, non statement**

A differenza di if, il ternario è un'**espressione**: ha un valore che puoi usare direttamente. Puoi assegnarlo, passarlo come parametro, usarlo in calcoli.**Quando usarlo**

Il ternario è elegante per assegnazioni semplici condizionali. Ma se la logica diventa complessa, un if normale è più leggibile. Il ternario dovrebbe migliorare la chiarezza, non oscurarla.