Modulo 12 - Le Eccezioni in Java

Un'eccezione rappresenta un'interruzione del normale flusso di esecuzione di un programma, causata da un'istruzione che non ha potuto completare correttamente il proprio compito. Questo meccanismo nasce generalmente dall'invocazione di un metodo che, per qualche ragione, non riesce a portare a termine l'operazione richiesta. Quando un metodo può generare un'eccezione, si dice che è un metodo "caldo", che richiede particolare attenzione nella gestione.

Consideriamo il caso di un metodo che legge un numero intero dall'input dell'utente. Se l'utente inserisce un valore che non può essere convertito in numero, il metodo non può completare la conversione e deve segnalare questo problema. La segnalazione avviene proprio attraverso la generazione di un'eccezione, che è un oggetto contenente tutte le informazioni rilevanti sull'errore verificatosi.

Propagazione e Gestione

Di fronte a una situazione eccezionale, un metodo si trova davanti a una scelta fondamentale: può decidere di propagare l'eccezione verso il chiamante oppure può gestirla direttamente al proprio interno. Questa decisione segue il principio di competenza, secondo cui un metodo dovrebbe gestire un'eccezione solo se dispone delle informazioni necessarie per farlo in modo appropriato. Se il metodo non ha gli elementi per decidere come procedere, la scelta migliore è notificare il problema al chiamante, che presumibilmente ha una visione più ampia del contesto.

Nel caso della lettura di un numero dall'input, il metodo che effettua la conversione non può sapere cosa fare se la conversione fallisce. Potrebbe essere necessario ripetere la domanda all'utente, utilizzare un valore predefinito, o terminare l'operazione. Questa decisione spetta al chiamante, quindi il metodo propaga l'eccezione. Il chiamante, che potrebbe essere il metodo principale del programma, dovrà invece gestirla, decidendo concretamente come reagire all'errore.

Quando un'eccezione viene generata e non viene gestita da nessun sottoprogramma lungo tutta la catena di chiamate, il risultato finale è il crash del programma. Questo sottolinea l'importanza di una corretta gestione delle eccezioni a qualche livello della gerarchia di chiamate.

Il Costrutto Try-Catch

La gestione delle eccezioni avviene attraverso una struttura try-catch, che presenta alcune analogie con il costrutto if-else. Il blocco try contiene il codice che esegue l'operazione potenzialmente problematica. Se tutto procede correttamente, l'intero blocco try viene eseguito e il blocco catch viene saltato. Se invece si verifica un'eccezione del tipo specificato nel catch, l'esecuzione del blocco try si interrompe nel punto esatto in cui l'eccezione è stata generata e il controllo passa immediatamente al blocco catch corrispondente.

La sintassi del catch richiede di specificare il tipo di eccezione da intercettare e un nome di variabile che rappresenterà l'oggetto eccezione. Questo perché l'eccezione non è una semplice segnalazione di errore, ma un vero e proprio oggetto che incapsula tutti i dettagli rilevanti dell'evento eccezionale. L'oggetto contiene informazioni come il messaggio di errore, la posizione nel codice dove si è verificato il problema e l'intera sequenza di chiamate a metodo che ha portato all'errore.

Informazioni sull'Eccezione

Quando un metodo dichiara di poter generare un'eccezione, questa dichiarazione diventa parte della sua interfaccia. Il metodo informa esplicitamente i suoi potenziali chiamanti che potrebbe verificarsi una determinata situazione eccezionale e che questa andrà gestita. L'oggetto eccezione mette a disposizione diversi metodi utili per comprendere cosa è accaduto. Il metodo getMessage fornisce una descrizione sintetica dell'errore, mentre printStackTrace ricostruisce l'intera catena di chiamate che ha portato all'eccezione, uno strumento prezioso durante la fase di sviluppo e debugging.

La stack trace, ovvero l'elenco ordinato delle chiamate a metodo che hanno condotto all'errore, permette di tracciare il percorso dell'esecuzione e identificare esattamente dove e perché si è verificato il problema. Durante lo sviluppo, è

pratica comune inserire printStackTrace in tutti i blocchi catch per avere una visione completa dell'accaduto.

La Parola Chiave Throws

La propagazione di un'eccezione viene dichiarata attraverso la parola chiave throws, che si colloca dopo la firma del metodo. Questa dichiarazione avvisa esplicitamente il chiamante che l'invocazione di quel metodo potrebbe generare quella particolare eccezione. Un metodo può propagare quante eccezioni desidera, elencarle tutte nella clausola throws, e può anche gestirne alcune internamente mentre ne propaga altre.

Non tutte le eccezioni devono essere obbligatoriamente gestite o propagate in modo esplicito. Le eccezioni si dividono in due categorie: checked e unchecked. Le eccezioni checked richiedono una gestione o propagazione esplicita obbligatoria. Se si invoca un metodo che può generare un'eccezione checked, il compilatore impone di racchiudere la chiamata in un blocco try-catch oppure di propagare a propria volta l'eccezione con throws. Le eccezioni unchecked, invece, hanno una gestione opzionale e una propagazione implicita. Non siamo costretti a dichiararle o gestirle esplicitamente, anche se possiamo farlo se lo riteniamo opportuno.

Gestione di Eccezioni Multiple

Un singolo blocco try può potenzialmente generare diverse tipologie di eccezioni, ciascuna delle quali potrebbe richiedere una gestione specifica. In questi casi, è possibile associare al blocco try molteplici blocchi catch, ognuno dedicato a un tipo particolare di eccezione. Quando si verifica un'eccezione, i blocchi catch vengono valutati in sequenza e il primo blocco che riconosce il tipo dell'eccezione la "consuma", eseguendo il proprio codice di gestione e saltando tutti i catch successivi.

Consideriamo un ciclo che elabora una sequenza di stringhe rappresentanti numeri. Il codice potrebbe fallire in due modi distinti: potrebbe incontrare una stringa che non rappresenta un numero valido, oppure potrebbe trovare un valore nullo. Questi due casi richiedono gestioni diverse, quindi definiamo due blocchi catch separati. Il primo intercetta gli errori di conversione numerica, il secondo

gestisce i riferimenti nulli. Il programma continua la propria esecuzione dopo aver gestito l'eccezione, invece di terminare bruscamente.

L'Ordine dei Blocchi Catch

L'ordine in cui vengono scritti i blocchi catch non è arbitrario ma segue una regola precisa: si parte dalle eccezioni più specifiche per arrivare a quelle più generiche. Questo perché le classi di eccezione sono organizzate in una gerarchia attraverso l'ereditarietà, e il primo blocco catch compatibile con il tipo dell'eccezione la consumerà. Se collocassimo un catch per un'eccezione generica prima di uno per un'eccezione specifica che ne deriva, il secondo catch non verrebbe mai raggiunto, costituendo codice morto che il compilatore rifiuterà di accettare.

Quando un'eccezione di tipo più specifico estende un'eccezione più generica, ogni istanza del tipo specifico è anche un'istanza del tipo generico per il principio di sostituzione. Se ordiniamo correttamente i catch dal più specifico al più generico, garantiamo che ogni eccezione venga intercettata dal blocco più appropriato. Un catch per Exception, che è l'antenato comune di tutte le eccezioni, dovrebbe sempre trovarsi in fondo, fungendo da rete di sicurezza per qualsiasi eccezione non gestita dai catch precedenti.

Multi Catch

Quando diverse tipologie di eccezioni richiedono la stessa gestione, Java permette di utilizzare un unico blocco catch per intercettarle tutte, separando i tipi con il simbolo della barra verticale. Questa sintassi abbreviata rende il codice più conciso ed evita duplicazioni, permettendo di raggruppare eccezioni anche se non sono correlate tra loro nella gerarchia di ereditarietà.

Generazione di Eccezioni

Oltre a gestire eccezioni generate da metodi altrui, è possibile generare eccezioni proprie utilizzando la parola chiave throw. Mentre throws dichiara che un metodo potrebbe produrre un'eccezione, throw effettivamente crea e propaga un oggetto eccezione che risalirà la catena di chiamate fino al primo blocco try-catch che lo intercetta.

Consideriamo un metodo che calcola valori della sequenza di Fibonacci. Se questo metodo riceve un parametro negativo, che non ha senso nel contesto matematico della sequenza, la risposta più logica è generare un'eccezione. Quando la condizione di errore si verifica, il metodo crea un nuovo oggetto eccezione, lo popola con un messaggio descrittivo, e lo lancia verso il chiamante. La generazione di un'eccezione termina immediatamente l'esecuzione del metodo, senza fornire un valore di ritorno normale. L'eccezione diventa una sorta di ritorno alternativo che comunica l'impossibilità di completare l'operazione.

Anche per le eccezioni generate manualmente vale la distinzione tra checked e unchecked. Se si genera un'eccezione checked, si è obbligati a dichiararla nella clausola throws del metodo. Per le eccezioni unchecked questa dichiarazione è opzionale, anche se rappresenta comunque una buona pratica per documentare il comportamento del metodo.

Il Blocco Finally

La struttura try-catch può includere un blocco opzionale chiamato finally, che viene sempre posizionato dopo tutti i blocchi catch. Il codice contenuto nel finally viene eseguito in ogni circostanza: sia che l'esecuzione sia passata attraverso il blocco try senza eccezioni, sia che abbia attraversato uno dei blocchi catch per gestire un errore. Il finally viene eseguito perfino quando il blocco try o un blocco catch contiene un'istruzione return che terminerebbe il metodo.

Questa caratteristica rende il finally il luogo ideale per operazioni di pulizia che devono essere sempre eseguite, indipendentemente dal flusso di esecuzione. Chiudere file aperti, rilasciare risorse, ripristinare stati: tutte queste operazioni trovano la loro collocazione naturale nel blocco finally, garantendo che vengano sempre eseguite anche in presenza di errori o uscite premature dal metodo.

Il Principio di Competenza

La decisione su quale metodo debba gestire un'eccezione non è casuale ma segue una logica progettuale precisa. Un metodo che si occupa di un'operazione specifica, come aprire un file o convertire una stringa in numero, spesso non ha le informazioni contestuali necessarie per decidere come reagire a un fallimento. Non sa se l'operazione può essere ripetuta, se esistono alternative, o se l'errore è

fatale per l'applicazione. Questi metodi propagano correttamente l'eccezione verso l'alto.

I metodi a livelli più alti nella gerarchia delle chiamate, che orchestrano operazioni più complesse e hanno una visione d'insieme del processo, sono invece nella posizione ideale per gestire l'eccezione. Possono decidere se riprovare l'operazione, utilizzare valori alternativi, chiedere ulteriori informazioni all'utente, o terminare l'esecuzione in modo controllato. Questa distribuzione delle responsabilità mantiene il codice modulare e ogni componente concentrato sul proprio compito specifico.

Tutte le eccezioni dovrebbero trovare un gestore prima di raggiungere il metodo principale del programma, o essere gestite da esso come ultima istanza. Questo garantisce che nessuna eccezione rimanga non gestita, evitando crash improvvisi e permettendo al programma di reagire in modo controllato anche nelle situazioni più problematiche.