Esercitazione week 6 – 06/08/2021

Laura Martines

# Quali sono le possibili cause di un’eccezione?

Le eccezioni sono errori scaturiti da situazioni impreviste che avvengono durante l’esecuzione di un programma; non sono quindi legate ad errori di scrittura del codice in sé e per sé, in quanto quelli non permetterebbero la compilazione del codice stesso.

Possono essere generate da “sviste” dello sviluppatore (i così detti errori di utilizzo) che non prende in considerazione determinate situazioni che potrebbero avvenire durante l’esecuzione del programma, come un inserimento non corretto dei dati da parte dell’utente ad esempio. Normalmente si sconsiglia di gestire questa categoria di errori attraverso le eccezioni, in quanto sarebbe meglio gestirli attraverso la programmazione preventiva o difensiva, ossia cercando di evitare che tali situazioni avvengano mettendo dei controlli (delle guardie) nel codice stesso. Ad esempio, se l’utente (o il codice in qualche modo) cerca di accedere ad un elemento di un vettore che è al di fuori della sua lunghezza, si genera una eccezione, ma tale errore può essere gestito più semplicemente inserendo delle righe di codice che verifichino che l’indice inserito non sia maggiore della lunghezza del vettore (o minore di 0).

Oltre alle sviste dello sviluppatore, le eccezioni possono essere generate per situazioni che non sono prevedibili né dallo sviluppatore né dal cliente, come ad esempio se si chiede di accedere o di modificare un file per cui non si ha l’autorizzazione (o se il file è corrotto), se il dispositivo su cui si opera riscontra un problema hardware o se si cerca di utilizzare una risorsa attualmente non disponibile. Tali errori son detti di programma o di sistema, e sono quelli che realmente hanno la necessità di essere gestiti tramite le eccezioni.

# Descrivere il costrutto formale per la gestione di una eccezione fornendo un esempio.

Le eccezioni non sono altro che classi derivate da una sola classe base che è Exception.

Quando viene generata una eccezione, il programma ripercorre in salita lo stack delle chiamate alla ricerca di un “gestore” dell’eccezione, ossia di un qualche costrutto che sia in grado di gestirla. Se il programma arriva al main senza aver trovato nessun gestore per l’eccezione in corso o per eccezioni più generiche (che quindi la possa comunque gestire), il programma viene interrotto bruscamente, se invece viene trovato un gestore per l’eccezione, esso la prende in carico per gestirla come previsto dallo sviluppatore.

Il gestore delle eccezioni è un blocco try-catch-finally, ossia in realtà un susseguirsi di blocchi. Nel blocco try viene inserito il codice che può generare eccezioni, quindi il programma prova ad eseguirlo; nel blocco catch si specifica sia l’eccezione che si vuole “catturare” (nel caso in cui venga generata dal codice inserito nel blocco try), sia il codice che specifica come cercare di porre rimedio all’eccezione nel caso in cui si genera. Infine, il blocco finally, che è opzionale, contiene del codice che verrà comunque eseguito sia nel caso in cui si generi l’eccezione che nel caso opposto. Tale blocco viene solitamente utilizzato nei casi in cui si lavori in comunicazione con qualche altro sistema (che può essere un database, oppure un file in lettura o scrittura, ad esempio), quando quindi si ha la necessità di chiudere la connessione per riportare una situazione di stabilità anche quando l’eccezione viene generata. Ultimamente, con il diffondersi dell’uso del blocco using, che chiude autonomamente la connessione quando finisce, l’uso del blocco finally sta diventando meno comune.

È possibile avere più blocchi catch in cascata in un solo try-catch, per catturare più errori distinti e dare informazioni più dettagliate all’utente riguardo alle righe di codice che hanno sollevato l’eccezione. In tal caso, è necessario inserire per prime le eccezioni più specifiche, mentre lasciare in fondo quelle più generali… in caso contrario, infatti il codice entrerebbe sempre in quelle più generali e mai in quelle più specifiche.

Esempio:

try

{

Array[] vettore = new array[4];

Int posizione = 5;

Array(posizione)=-3;

}

Catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

In questo caso, dato che la posizione 5 non esiste nell’array vettore (che ha quattro elementi, numerati da 0 a 3), quando si cercherà di modificare il valore in posizione 5 verrà scatenata una eccezione che sarà catturata dal blocco catch (in cui è stata inserita l’eccezione base, e che quindi catturerà qualsiasi eccezione si verifichi). Il gestore a questo punto stamperà a video il messaggio di errore della specifica eccezione (che è una proprietà dell’eccezione stessa), in modo da aiutare l’utente a capire quale errore si sia verificato. In questo specifico caso, si sarebbe potuto risolvere usando la programmazione difensiva.

# Descrivere i ruoli che esistono all’interno di un progetto

All’interno di un progetto, i ruoli principali sono 6, anche se non sempre sono tutti presenti (i primi due sono clienti, gli altri membri del team di sviluppo vero e proprio):

1. Product owner: colui che commissiona l’applicazione software.
2. Business analysts: sono figure di ponte tra il product owner e il team di sviluppo software vero e proprio. Possono essere rappresentanti degli utilizzatori finali dell’applicazione, che sanno come essa dovrà essere utilizzata e possono spiegare meglio al team software i requisiti necessari (il product owner potrebbe essere anche un business analyst)
3. Il software/solution architect: realizza concettualmente il design dell’architettura, identificando i modi in cui gli sviluppatori devono costruire il sistema. Il suo compito è delicato perché errori generati in questa fase possono diventare problematici in fase di sviluppo, se non rilevati in tempo l’architetto potrebbe essere anche uno sviluppatore)
4. Sviluppatori: coloro che scrivono il codice vero e proprio basandosi sui disegni funzionali e tecnici redatti dall’architetto e dagli analisti
5. Team Quality assurance 🡪 testa il codice per trovare eventuali falle nel sistema, e per assicurarsi che esso soddisfi i requisiti (potrebbero essere gli stessi sviluppatori, anche se non è consigliato)
6. Team operations 🡪 una volta che il software è finito e testato, lo installa per il cliente e crea gli ambienti di esecuzione temporanea e finali ( può coincidere con gli sviluppatori, con il team QA o con entrambi).

# Descrivere la differenza tra class diagram e object diagram descrivendo gli elementi di cui possono essere composti.

Il class diagram rappresenta le classi in gioco nell’applicazione (quindi gli oggetti statici, i nomi identificati dai requisiti del cliente) e le loro relazioni. Ogni classe ha un nome, può avere attributi (pubblici, privati, protected ecc) e metodi (le operations) e possono essere messe in relazione di associazione (comprensiva di nome dell’associazione e di … ), di generalizzazione (l’equivalente della ereditarietà) e di aggregazione/composizione. L’object diagram deriva dal class diagram ma contiene esempi di istanze di classe, con relativi esempi di valori che i loro attributi possono assumere. Serve a dare una idea pratica di come poi le singole istanze risulterebbero