APPROFONDIMENTO DI TPSI

JUnit 5

JUnit 5 è l'ultima versione del popolare framework di testing per Java.
È una riscrittura completa rispetto alle versioni precedenti (JUnit 3 e JUnit 4) e introduce diverse nuove funzionalità e miglioramenti.

JUnit 5 è composto da diversi moduli, tra cui:

- JUnit Platform: La base per l'avvio dei framework di testing su JVM.
- JUnit Jupiter: Il nuovo modello di programmazione e di estensione per scrivere test in JUnit 5.
- JUnit Vintage: Supporto per l'esecuzione di test scritti con JUnit 3 e JUnit 4.

Alcune delle caratteristiche principali di JUnit 5 includono:

- Annotations avanzate: Nuove annotazioni come @Test, @BeforeEach, @AfterEach, @BeforeAll, @AfterAll, @DisplayName, ecc.
- Supporto per lambda: Permette di scrivere test più concisi e leggibili.
- Estensioni: Un nuovo modello di estensione che permette di personalizzare il comportamento dei test.
- Test parametrici: Supporto per test che vengono eseguiti con diversi set di dati.
- Test Unitari: Testano singole unità (metodi o classi) del codice.
- Test di Integrazione: Testano l'interazione tra più componenti del sistema.
- Test di Accettazione: Testano la conformità del software ai requisiti.
- Test Parametrizzati: Permettono di eseguire lo stesso test con differenti input e verifiche.
- Test Basati su Eccezioni: Verificano che venga lanciata un'eccezione prevista in condizioni specifiche.
- Condizioni di test dinamici: Permette di creare test in modo dinamico durante l'esecuzione.

Test Driven Development

Il Test Driven Development (TDD) è una metodologia di sviluppo software che enfatizza la scrittura dei test prima della scrittura del codice effettivo. Il processo TDD segue un ciclo iterativo composto da tre fasi principali:

1. Red (Scrivere un test che fallisce): Si inizia scrivendo un test per una nuova funzionalità o correzione di bug. Questo test inizialmente fallirà perché la funzionalità non è ancora stata implementata.
2. Green (Scrivere il codice per far passare il test): Si scrive il codice minimo necessario per far passare il test. L'obiettivo è ottenere un test che passi, anche se il codice non è ancora ottimizzato o completo.
3. Refactor (Migliorare il codice): Una volta che il test passa, si ri-fattorizza il codice per migliorarne la qualità, mantenendo il test verde.

I vantaggi del TDD sono i seguenti:

- Migliore qualità del codice: I test aiutano a identificare e correggere i bug prima che diventino problemi più grandi.

- Documentazione vivente: I test servono come documentazione vivente del comportamento del codice.
- Design migliore: Scrivere test prima del codice incoraggia un design più modulare e disaccoppiato.
- Maggiore fiducia: Avere una suite di test completa aumenta la fiducia nel fare modifiche al codice senza introdurre regressioni.

Javadoc

La Javadoc è uno strumento di documentazione per il linguaggio di programmazione Java.

Permette di generare una documentazione HTML a partire dai commenti speciali inseriti nel codice sorgente. La documentazione generata aiuta gli sviluppatori a comprendere meglio il funzionamento delle classi, metodi e variabili, facilitando la manutenzione e l'estensione del software.

Per creare la Javadoc, bisogna aggiungere dei commenti nel codice utilizzando una sintassi speciale. Questi commenti vengono poi elaborati da un tool, che crea la documentazione in formato HTML.

I commenti Javadoc sono identificati con /** e terminano con */. All'interno di questi commenti si utilizzano tag specifici per descrivere classi, metodi e variabili.

Per generare la documentazione da riga di comando si utilizza il seguente comando:
javadoc -d doc nomefile.java
Tale comando genera la documentazione in una cartella specifica.

In Javadoc, è importante documentare i seguenti aspetti del codice:

- Classi: La descrizione della classe e del suo scopo.
- Costruttori: Descrivere cosa fa ogni costruttore e i parametri che riceve.
- Metodi: Descrivere cosa fa ogni metodo, i parametri che accetta e i valori che restituisce (se presenti).
- Variabili: Se necessario, commenta le variabili per spiegare il loro significato.
- Eccezioni: Se un metodo può generare eccezioni, dovresti descrivere quali eccezioni possono essere sollevate e in quali condizioni.

La Javadoc inoltre supporta l'uso di tag HTML per arricchire la documentazione con formattazione aggiuntiva, come liste, tabelle, link, etc.

Alcuni dei tag HTML più comuni sono:

- <b>: Grassetto
- <i>: Corsivo
- , <ol>, <li>: Liste non ordinate e ordinate
- <code>: Per il codice in linea
- <pre>: Per il codice preformattato
- <a href="url">: Per i link

Le best Practices nell'utilizzo della Javadoc sono:

- Documentare tutto ciò che è pubblico: Se una classe, un metodo o una variabile è accessibile pubblicamente, dovrebbe essere documentata.
- Descrivere chiaramente l'intento del codice: Evita descrizioni ovvie e concentrati sul comportamento e lo scopo delle entità.
- Utilizzare i tag correttamente: Usa i tag @param, @return, e @throws in modo coerente per tutti i metodi pubblici.
- Essere concisi ma chiari: Le descrizioni dovrebbero essere brevi, ma sufficientemente dettagliate da spiegare il comportamento senza dover guardare il codice.
- Aggiornare la Javadoc insieme al codice: Assicurati che i commenti siano aggiornati quando cambi il codice. Una documentazione obsoleta può essere più dannosa di nessuna documentazione.
- Evitare di documentare implementazioni interne: Se una classe o metodo è privato, generalmente non è necessario documentarlo (a meno che non ci siano ragioni speciali per farlo).

Espressioni regolari (RegExp)

Le espressioni regolari sono uno strumento per il matching e le manipolazioni di stringhe.

In informatica un'espressione regolare definisce una funzione che ha come parametro una stringa e restituisce un valore booleano a seconda che la stringa segua o meno un determinato pattern.

Tali espressioni iniziarono ad essere utilizzate alla fine degli anni sessanta negli ambienti Unix quando il matematico Stephen Kleene sviluppò la teoria degli automi e linguaggi formali.

Tali espressioni regolari sono suddivise in due categorie, BRE e ERE:

- BRE hanno sintassi più semplici, alcuni metacaratteri devono essere preceduti per questo da una sequenza di escape.
- ERE hanno invece sintassi più potente che supporta metacaratteri senza necessità di \.

I metacaratteri sono dei caratteri speciali che modificano il comportamento della ricerca RegExp.

I metacaratteri più diffusi sono i seguenti.

.	Qualsiasi carattere tranne newline	gr.p -> trova grep, grip, grap
^	Inizio della stringa	^Errore → Trova "Errore" solo all'inizio
$	Fine della stringa	log$ → Trova stringhe che terminano con "log"
*	Zero o più occorrenze	fo* → Trova f, fo, foo, fooo
+	Una o più occorrenze	fo+ → Trova fo, foo, fooo, ma non f
?	Zero o una occorrenza	colou?r → Trova color e colour
{n,m}	Minimo n, massimo m ripetizioni	[0-9]{2,4} → Trova numeri di 2-4 cifre

[]	Classe di caratteri	[aeiou] → Trova vocali
()	Raggruppamento	(abc){2} → Trova "abcabc"

Pipelining

Il pipelining è una tecnica in bash che permette di passare l'output di un comando come input a un altro comando utilizzando il simbolo | (pipe).

Il pipelining è utile per concatenare più comandi in modo efficiente senza dover salvare temporaneamente determinati output.

Esempio:

cat file.txt | wc -l

Con cat file.txt si stampa il contenuto del file e tale output va come input a wc -l esso serve per contare le righe.

Grep

Il comando grep (Global Regular Expression Print) è utilizzato per cercare un testo specifico all'interno di file o output di altri comandi.

La sintassi di tale comando è la seguente: *grep [opzioni] "pattern" file*

Delle opzioni comuni per il comando grep sono:

-i	Ignora maiuscole e minuscole
-v	Mostra le righe che non contengono il pattern
-c	Conta il numero di righe che contengono il pattern
-n	Mostra il numero di riga prima di ogni corrispondenza
-E	Usa espressioni regolari estese
-r	Cerca in modo ricorsivo nelle sottodirectory
-l	Mostra solo i nomi dei file che contengono il pattern
-w	Cerca parole intere
-o	Stampa solo la parte della riga che corrisponde al pattern

Combinare tale comando con le espressioni regolari risulta molto utile per cercare dei pattern specifici e complessi, un esempio può essere il seguente:

grep "^Errore" logfile.txt

Esso cercherà tutte le righe che iniziano con la stringa "Errore".

Find

Il comando find viene utilizzato per cercare file e directory in modo avanzato all'interno di un percorso specificato.

La sintassi di tale comando è la seguente: *find [percorso] [opzioni] [criteri]*

Il percorso definisce la directory da cui iniziare la ricerca, le opzioni modificano il comportamento del comando e infine i criteri definiscono il tipo di ricerca da effettuare.

Delle opzioni comuni per il comando grep sono:

-name "nome"	Cerca un file per nome (case-sensitive)
-iname "nome"	Cerca un file per nome (ignora maiuscole/minuscole)
-type f	Cerca solo file
-type d	Cerca solo directory
-mtime -7	Cerca file modificati negli ultimi 7 giorni
-atime +30	Cerca file non aperti negli ultimi 30 giorni
-user utente	Cerca file appartenenti a uno specifico utente
-exec <comando> {} \;	Esegue un comando su ogni file trovato
-delete	Cancella direttamente i file trovati
-empty	Cerca file vuoti

Il comando find supporta le espressioni regolari tramite l'utilizzo dell'opzione *-regex*, un esempio può essere:

find /var/log -type f -regex ".\.\(txt\|log\)$"*

Tale comando cerca i file con estensione .txt o .log all'interno della cartella /var/log, l'espressione regolare indica che bisogna filtrare qualsiasi nome e percorso e che contengono alla fine la stringa . seguita da txt o log.

Numeri non interi bash

I numeri in virgola mobile possono rappresentare una gamma molto ampia di valori con diversi gradi di precisione. Tuttavia, Bash non ha una sintassi nativa per i calcoli in virgola mobile di conseguenza, è necessario ricorrere a strumenti esterni quali:

- Basic calculator: esso è un linguaggio di calcolo orientato agli oggetti che fornisce un'interfaccia a passo interattivo per eseguire calcoli matematici. Supporta operazioni sia in virgola mobile che in interi, offrendo funzionalità avanzate come variabili e funzioni.

- Awk: è un linguaggio di programmazione per il processamento di testi, ma ha anche capacità matematiche. È molto utile per analizzare file di testo e per effettuare calcoli, in particolare quando si gestiscono dati estratti da file.

Le operazioni eseguibili sono:
- Somma
- Sottrazione
- Moltiplicazione
- Divisione

Esempio in basic calculator :
echo "2.3 + 3.7" | bc
Esempio in Awk:
awk "BEGIN {print 2.3 + 3.7}"