Iterazione 3

Tra i casi d'uso a priorità maggiore rimane l'UC4 da codificare.

Nome	ID	Tipo	Priorità	Descrizione	Req. padre	Req. figli
GENERA TRANSAZIONE	UC4	funzionale	3	Se l'utente ha accettato la proposta, il sistema genera una transazione e invia i dati ai server di Ethereum per la scrittura in un blocco della Blockchain.	UC3	UC2

Il punto chiave di questo caso d'uso è l'utilizzo di un algoritmo che generi e invii una transazione.

Prima devo definire cosa sia una transazione: *un Proposal che passa in stato di CONFIRMED genera una Transaction, che contiene i dati che l'utente vuole scrivere in un nodo della blockchain.*

Il sistema può accedere alla scrittura su un nodo della blockchain una transazione alla volta. Di conseguenza è necessario progettare un algoritmo che gestisca l'ordine di scrittura delle transazioni su un nodo.

Ad ogni transazione associo una metrica che dipende dalla lunghezza del campo String description. Questo in prima approssimazione. In future iterazioni potrebbero essere integrati l'upload di files, dunque in quel caso farei dipendere la metrica dalla dimensione in bytes degli allegati.

In ogni caso, è necessario associare un elemento temporale ad ogni transazione tramite cui ordinarle. L'obiettivo è quello di **ottimizzare il tempo di scrittura medio**.

Se n è il numero totale di transazioni e ad ogni transazione i-esima associo un tempo di scrittura T(i), il tempo di scrittura medio vale:

$$T_{AVG} = \frac{T}{n} = \frac{1}{n}\sum_{i=1}^{n} T(i)$$

Il tempo totale di scrittura è dato dalla somma dei tempi.

Scelta **greedy**: prendiamo al j-esimo passo la transazione più veloce tra quelle da scrivere.

Pseudo codice

ALGORITMO SCRITTURA(Array C) → Soluzione:

    S = { }
    C = tempi(durata) t; per j=1 … n delle scritture richieste
    Ordina C in modo non decrescente //Preprocessing

```
FOR j=1 TO n DO
        S = S U {j};
RETURN s;
```

Calcolo complessità

La cardinalità dell'array in ingresso all'algoritmo è |C|=n.
Le istruzioni rilevanti sono:

- l'ordinamento, che nel Java Collection è pseudo-lineare: O(n logn)
- il ciclo for esegue n iterazioni: O(n)
- la scelta greedy all'interno del ciclo costa O(1): è un semplice assegnamento.

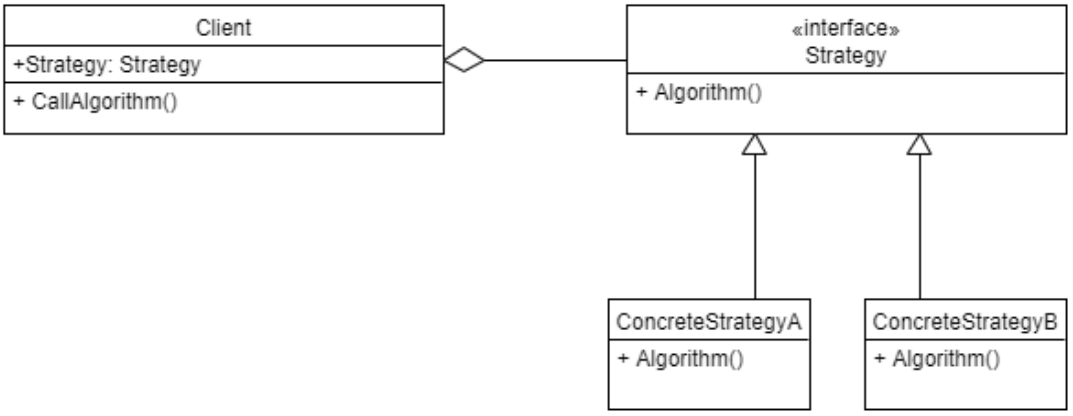
Il costo complessivo è:

$$T(n) = O(n\,logn) + O(n) = O(n\,logn)$$

Implementazione dell'algoritmo

Per implementare l'algoritmo uso il pattern Strategy:

- se cambia l'implementazione, non devo ritoccare tutti i punti del codice in cui usiamo l'algoritmo.
- può succedere che diverse implementazioni vengano scelte dal programma.



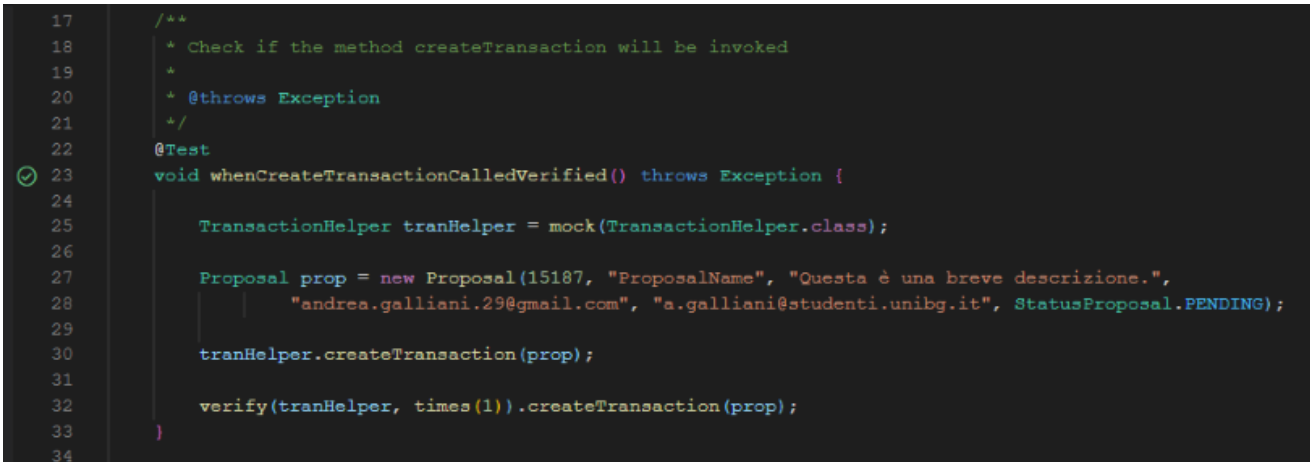
1. definisco l'interfaccia PriorizerGreedy
2. introduco la classe TransactionWriter che implementa l'interfaccia e fa override dell'algoritmo.
3. Testo l'algoritmo creando un oggetto consumatore. Dichiaro un oggetto che ha per tipo l'interfaccia e usa l'operatore di upcasting new, a cui è delegata la scelta dell'algoritmo.

Test

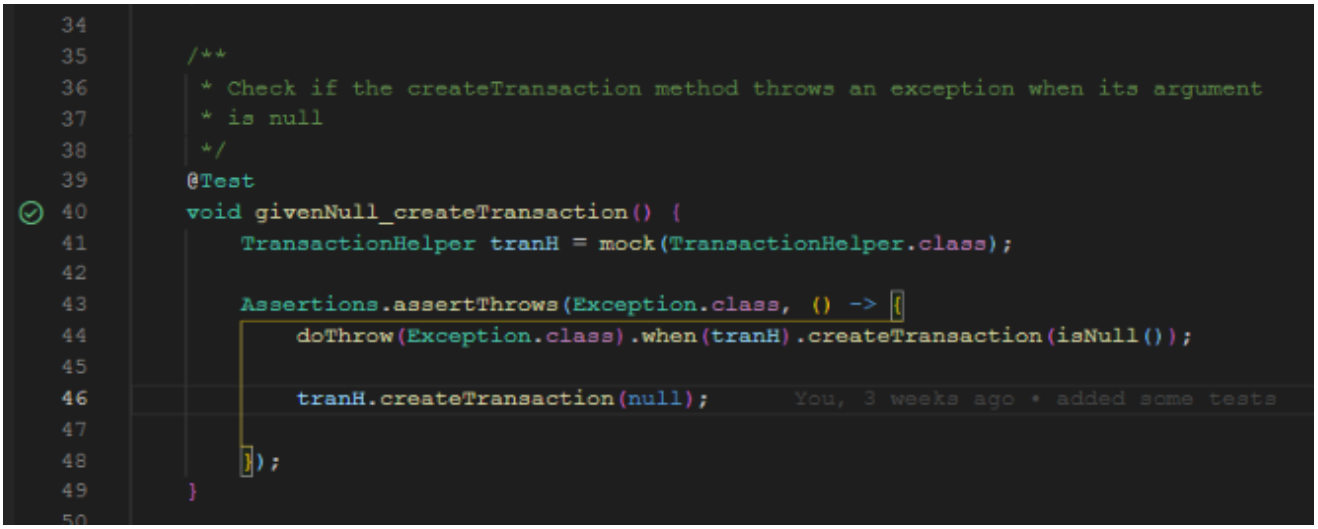
Ho deciso di testare il componente TransactionHelper, perchè è quello che si occupa di gestire la creazione e l'invio di un oggetto Transaction.

Utilizzo gli strumenti di test messi a disposizione dal framework Spring Boot e vado a verificare due casi:

1. la creazione di una Transaction a seguito della conferma di una Proposal. Verifico che effettivamente il metodo *createTransaction* venga invocato



2. verico che a seguito di un argomento nullo in input al metodo *createTransaction* il sistema lancia un'eccezione



Altri due casi di test riguardano l'invocazione del metodo di invio a Ethereum. Il cui server è mockato.

```java
    /**
     * Check if the method is correctly invoked
     */
    @Test
    void verifySendToEthereum() {

        TransactionWriter transactionWriter = mock(TransactionWriter.class);

        TransactionHelper tHelper = new TransactionHelper();

        Proposal prop = new Proposal(15187, "ProposalName", "Questa è una breve descrizione.",
                "andrea.galliani.29@gmail.com", "a.galliani@studenti.unibg.it", StatusProposal.PENDING);

        Transaction tran = tHelper.buildTransaction(prop);

        transactionWriter.sendToEthereum(tran);

        verify(transactionWriter).sendToEthereum(tran);

    }
```

Infine, testo che effettivamente l'operazione greedy venga eseguita. Ovvero che la scrittura avvenga in ordine crescente rispetto la metrica calcolata proporzionalmente alla dimensione dei dati.

```java
    @Test
    void testWritingOrderer() {

        List<Transaction> elenco_transazioni = new ArrayList<>();

        TransactionHelper tHelper = new TransactionHelper();

        // popolo la lista di transazioni
        for (int i = 0; i < 10; i++) {
            elenco_transazioni
                    .add(tHelper.buildTransaction(new Proposal(15187, "ProposalName", "Questa è una breve descrizione.",
                            "andrea.galliani.29@gmail.com", "a.galliani@studenti.unibg.it", StatusProposal.PENDING)));
        }

        Collections.shuffle(elenco_transazioni);

        TransactionWriter tWriter = mock(TransactionWriter.class);
        tWriter.writingOrderer(elenco_transazioni);
        verify(tWriter).writingOrderer(elenco_transazioni);


    }
}
```