ids-cheatsheet

title: "Ingegneria Del Software - CheatSheet"
author:

- "Nadav Moscovici"

Ingegneria del software

Functional Programming

Metodo	Descrizione	Ritorna	Note/Esempio
allMatch(Predicate<? super T> predicate)	Controlla se tutti gli elementi dello stream corrispondano all'predicato fornito	Ritorna un booleano	allMatch(n -> n%3==0)
anyMatch(Predicate<? super T> predicate)	Controlla se almeno un elemento dello stream corrisponde all'predicato fornito	Ritorna un booleano	anyMatch(n->(n*(n+1))/4 == 5)
collect(Collector<? super T,A,R> collector)	Ritorna una collezione <R, A> R dopo aver eseguito un'operazione di riduzione mutabile sugli elementi dello stream usando un collector	Ritorna un booleano	collect(Collectors.toList())
concat(Stream<? extends T> a, Stream<? extends T> b)	Concatena lo Stream b allo Stream a	Ritorna uno Stream<T>	concat(stream1, stream2)
count()	Conta il numero di elementi presenti nello Stream	Ritorna un long	stream().count()
distinct()	Tiene solo gli elementi diversi tra di loro	Ritorna uno Stream<T>	stream().distinct()
empty()	Crea uno stream sequenziale vuoto	Ritorna uno Stream<T>	stream = Stream.empty()
filter(Predicate<? super T> predicate)	Prende solo gli elementi dello stream	Ritorna uno Stream<T>	filter(n -> n>=18)

	che rispettano il predicato		
findAny()	Descrive un elemento qualsiasi dello stream	Ritorna un Optional<T>	**stream().findAny()** *ritorna un opzionale vuoto se lo stream è vuoto*
findFirst()	Descrive il primo elemento dello stream	Ritorna un Optional<T>	**stream().findFirst()** *ritorna un opzionale vuoto se lo stream è vuoto*
flatMap(Function<? superT, ? extends Stream<? extends R>> mapper)	Mette tutti i "subelementi" di ogni elemento nello stream originale uno dopo l'altro	Ritorna uno Stream<T>	**flatMap(g -> g.getName().stream())**
forEach(Consumer<? super T> action)	Compie un azione su ogni elemento dello stream	Ritorna void	**forEach(System.out::println)**
limit(long maxSize)	Tronca lo stream originale alla lunghezza maxSize fornita	Ritorna uno Stream<T>	**limit(3)** **limit(range)**
max(Comparator<? super T> comparator)	Ritorna l'elemento massimo dello stream che rispetta il comparatore fornito	Ritorna un Optional<T>	**max(Integer::compareTo)** *Questo ritorna in base all'ordine naturale degli interi*
min(Comparator<? super T> comparator)	Ritorna l'elemento minimo dello stream che rispetta il comparatore fornito	Ritorna un Optional<T>	**min(Comparator.reverseOrder())** *Questo ritorna in base all'inverso dell'ordine naturale degli interi (quindi ritorna il massimo)*
noneMatch(Predicate<? super T> predicate)	Dice se nessuno elemento dello stream corrisponde al predicato fornito	Ritorna un booleano	**noneMatch(str -> (str.length() == 4))**
of(T... values)	Crea uno stream sequenziale ordinato dove gli elementi sono i valori passati	Ritorna uno Stream<T>	**stream = Stream.of("CSE", "C++", "Java", "DS")**
reduce(BinaryOperator<T> accumulator)	Esegue una riduzione sugli elementi dello stream usando una funzione di accumulazione associativa ritorna	Ritorna un Optional<T>	**reduce((word1, word2) -> word1.length() >**

	poi il risultato della riduzione se presente		**word2.length() ? word1 : word2)**
skip(long n)	**Salta i primi n elementi dello stream originale**	**Ritorna uno Stream<T>**	**skip(5)** **skip(jump)**
sorted(Comparator<? super T> comparator)	**Ritorna uno stream formato dagli elementi dello stream originale ordinati in base al comparatore fornito**	**Ritorna uno Stream<T>**	**sorted(Comparator.reverseOrder())** ***Se non viene fornito alcun comparatore ordina in base all'ordine naturale***
toArray()	**Ritorna un array contente tutti gli elementi dello stream**	**Ritorna un Object[]**	**stream().toArray()**

Map

Interface Map<K,V>

Dove K è il tipo della chiava della mappa e V è il tipo dei valori mappati

Ex:

```
//questa è una mappa che usa un Double come chiave e ogni valore è rappresentato da una stringa
Map<Double, String> utenti;

//in questo caso la mappa usa una stringa come chiave e i valori sono rappresentati da una mappa
//che usa un tipo Subject come chiave e i valori sono rappresentati da una lista di Double
Map<String, Map<Subject, List<Double>>> studenti;
```

Metodi di Map

> void clear(): cancella tutti i valori e le chiavi della mappa. La mappa sarà quindi vuota dopo che questo metodo ha terminato.
>
> int size(): Ritorna il numero di associazioni chiave-valore presenti in questa mappa.

boolean containsKey(Object key): Ritorna vero se questa mappa contiene una cella per la chiave specificata.

boolean containsValue(Object value): Ritorna vero se la mappa associa ad una o piu chiavi il valore specificato.

Set<Map.Entry<K, V>> entrySet(): Ritorna un set contenente tutte la mappature presenti nella mappa. Questo set è costruito sulla mappa, quindi cambiamenti futuri alla mappa si rifletterano sul Set e vice-versa.

V get(Object key): Ritorna il valore associato alla chiave specificata, ritorna null se non c'è alcun valore associato alla chiave.

V put(K key, V value): Associa il valore passato alla chiave specificata. Se la mappa gia conteneva un valore associato a quella chiave, il nuovo valore sovrascriverà quello vecchio.

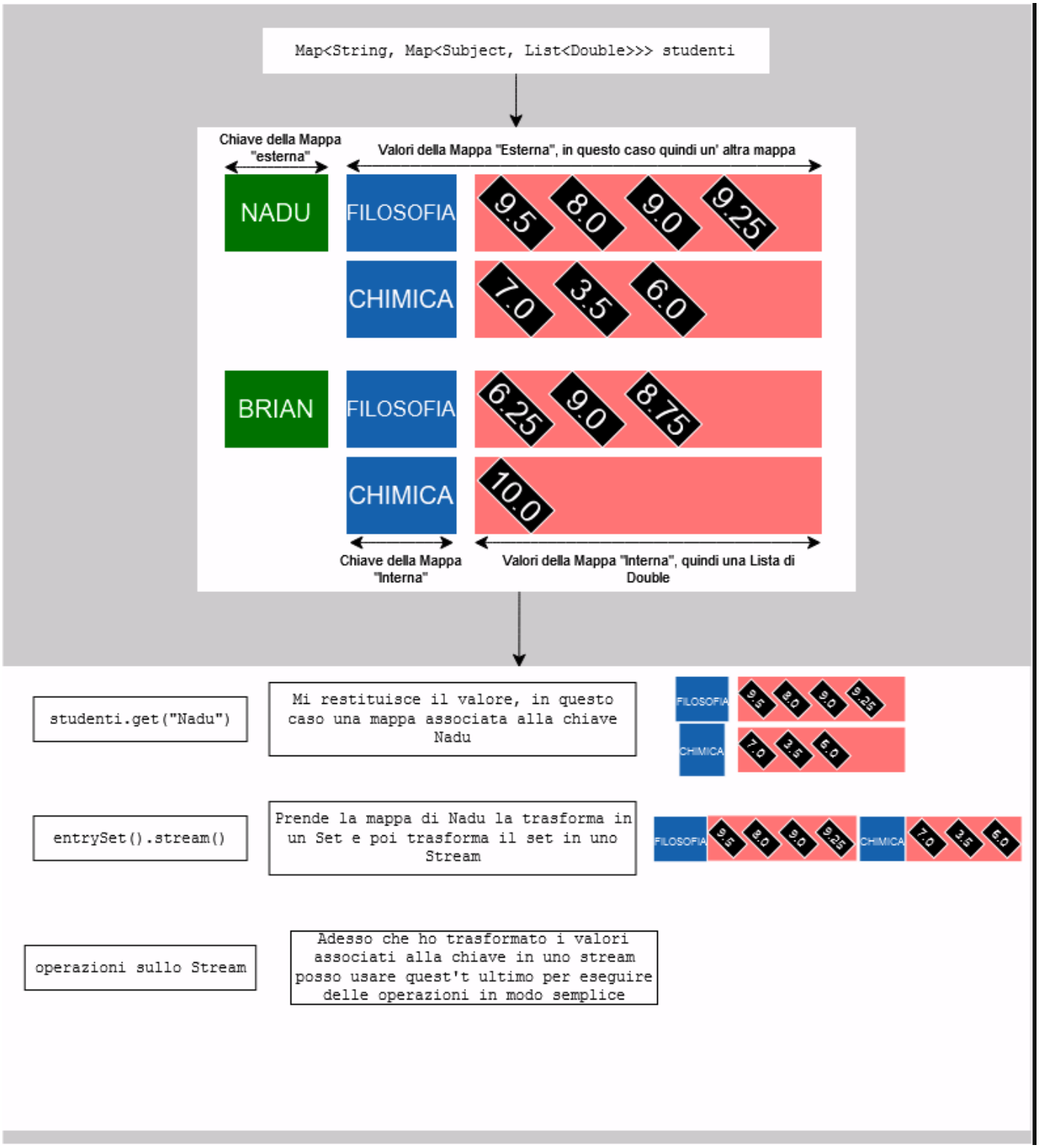
V replace(K key, V oldValue, V newValue): Sostituisce il valore associato alla chiave passata con newValue solo se il valore attuale è oldValue. Se oldValue viene omesso, allora gli va bene qualsiasi valore basta che non sia null o mancante.

boolean remove(Object key, Object value): Rimuove la cella associata alla chiave specificata solo se associata al valore passato.

boolean isEmpty(): Ritorna vero se la mappa non contiene associazioni chiave-valore

Set keySet(): Ritorna un set con tutte le chiavi contenute nella mappa. Questo Set è costruito sulla

> mappa, quindi futuri cambiamenti alla mappa si rifletterano sul Set e vice-versa.
>
> Collection values(): Ritorna una collezione di tutti i valori contenuti nella mappa. La collezione è costruita sulla mappa, quindi futuri cambiamenti alla mappa si rifletterano sulla collezione e vice-versa.



Ex:

```
studenti.get("Nadu").entrySet().stream().map(
    (entry) -> {
```

```
        return new Pair<Subject, Double>(
            entry.getKey(), (
                entry.getValue().stream().reduce(0.0, (a, b) ->
a+b)/entry.getValue().size()
                )
            );
    }
).collect(Collectors.toList());

//Questo restituisce la media per ogni materia dello studente
//attenzione che entry è un Pair non una mappa
//i metodi getKey() e getValue() non appartengono a map e non possono essere
usati su una mappa
//Un pair non è una mappa, ma indica solo una coppia di valori che vogliamo
tenere assieme
//Pair ha i seguenti metodi:
// String toString(): questo metodo restituirà la rappresentazione String
della coppia.
// K getKey(): Restituisce la chiave per la coppia.
// V getValue(): Restituisce un valore per la coppia.
// int hashCode(): genera un codice hash per la coppia.
```

Synchronized and Multi-Threading

Synchronized

Non è possibile che due invocazioni di due metodi sincronizzati sullo stesso oggetto si sovrappongano.

Quando un Thread sta eseguendo un metodo sincronizzato su un oggetto tutti gli altri thread che invocano metodi sincronizzati
sullo stesso oggetto vengono bloccati fino a che il primo thread non ha finito con l'oggetto.

Quando esiste un metodo sincronizzato, questo stabilisce automaticamente una relazione di *happens-before* (succede-prima)
con tutte le invocazioni successive del metodo sincronizzato sullo stesso oggetto.

Questo garantisce che lo stato dell'oggetto sia visibile a tutti i thread.

```
public synchronized void increment()
{
    randomString = "TiTilda";
    c++;
}
```

Qua sincronizzo tutto il metodo, quidni fino a che il metodo non ha finito, l'oggetto è bloccato

```
public void increment()
{
    randomString = "TiTilda";

    synchronized(this)
    {
        c++;
    }
}
```

In questo caso randomString non è una variabile condivisa quindi non ho bisogno di modificarla in un momento sincronizzato
Invece c è condiviso quindi la modifico sincronizzando il metodo.
Alternativamete al posto di 'this' potevo mettere un oggetto di lock, cosi sincronizzavo sull'oggetto e non sul metodo

sleep(long amount)

Blocca il thread per una quantità di tempo specifica, definita in milisecondi, prima di farlo ripartire

```
synchronized public void sing()
{
    print("She Drives Me Crazy");
    sleep(3500);
    print("Like No One Else");
}
```

wait()

Blocca il thread per un tempo indefinito e lo inserisce in una coda fino a quando questo non viene risvegliato

```
synchronized public void shoot()
{
    if(bulletsLeft == 0) wait();
    else bulletsLeft--;
}
```

notify()

Risveglia il primo thread della coda premettensogli di riprendere la sua esecuzione

```
synchronized public void reload()
{
    bulletsLeft = maxBullets;
```

```
    notify();
}
```

notifyAll()

Risveglia tutti i thread della coda permettendo a tutti di riprendere la loro esecuzione

```
public void run()
{
    print("Notifying All");
    synchronized(this)
    {
        notifyAll();
    }
    print("All Notified");
}
```

JML

Espressione	Descrizione	Tipo	Note
\result	Puo essere usato solo nell'ensures di un metodo non-void. Ilsuo valore e tipo sono quelli ritornati dal metodo	? (*qualsiasi tipo*)	
\old(val)	Indica il valore che val aveva nella precondizione	?	
\not_assigned(val-list)	Puo esseren usata nell'ensures e nella signals, indica che i valori val-list non sono mai stati assegnati	booleano	i valori di val-list sono forniti come parametri in un metodo separati dalla virgola
\not_modified(val-list)	Puo essere usata nell'ensures e nella signals, indica che i valori non sono cambiati rispetto alla pre-condizione	booleano	
\only_accessed(val-list)	Puo essere usata nell'ensures e nella signals,indica che nessun valore al di fuori di quelli di val-list è stato letto dal metodo	booleano	

\only_assigned(val-list)	**Puo essere usata nell'ensures e nella signals, indica che nessun valore al di fuori di quelli di val-list è stato assegnato dal metodo**	**booleano**	
\only_called(method-list)	**Puo essere usato nell'ensures e nella signals, indica che nessun metodo al di fuori di quelli di method-list è stato chiamato**	**booleano**	
\fresh(val-list)	**Puo essere utilizzato nell'ensures e nella signals, indica che i valori di val-list non sono stati inizializzati nella pre-condizione**	**booleano**	
\nonnullelements(val)	**controlla che tutti gli elementi dell'array o della collezione siano non nulli**	**predicato (*boolean like*)**	**è l'equivalente di fare:** ***myArray != null && (\forall int i; 0 <= i="" &&="" <="" myarray.length;="" myarray[i]="" !="null) </i" class="jop-noMdConv">;***
\typeof(val)	**Ritorna il tipo di val**	**?**	**Per esempio posso fare:** ***\typeof(num) == Integer*** **per controllare se num è un intero**
\max(val)	**Ritorna il massimo dell'set passato**	**?**	**Se ho un array devo prima transformarlo in set:** ***\max(\set(i; 0 <= i="" &&="" <="" arr.length;="" arr[i])) <="" class="jop-noMdConv">;***
\min(val)	**Ritorna il minimo dell'set passato**	**?**	
\forall(var; condizione; predicato)	**Controlla che tutti gli elementi nell'range della condizione rispettino il predicato**	**quantificatore (*boolean like*)**	

\exists(var; condizione; predicato)	**Controlla che esista almeno un valore nell'range della condizione rispetti il predicato**	**quantificatore (*boolean like*)**	
\num_of(var; condizione)	**Conta i valori in var che rispettino la condizione**	**long**	
\sum(var; condizione; val)	**fa la somma di tutti i val, cui var rispetta la condizione**	**?**	***\sum(int i; 0 <= i="" &&="" <="" arr.length;="" arr[i]) <="" class="jop-noMdConv">;***
\product(var; condizione; val)	**fa il prodotto di tutti i val, cui var rispetta la condizione**	**?**	***\product(int i; 0 <= i="" &&="" <="" arr.length;="" arr[i]) <="" class="jop-noMdConv">;***
\is_initialized(class)	**Controlla se la classe ha finito la sua inizializzalizazione statica**	**booleano**	
\choose(var; condizione; predicato)	**Ritorna un qualsiasi valore nel range della condizione che rispetti il predicato**	**?**	***\choose(int i; 0 <= i="" <="" a.length;="" a[i]="=" 0)<="" class="jop-noMdConv">;***
==> ***implica***	**Puo essere usato solo su sub-espressioni booleane A==>B puo essere tradotto in *!A || B***	**booleano**	**Puo anche essere scritto al contrario: A<==B, tradotto quindi in A || !B**
<==> ***se solo se***	**Puo essere usato solo su sub-espressioni booleane A<==>B puo essere tradotto in *A == B***	**booleano**	
<=!=> ***se solo se non*** **(*non equivale a*)**	**Puo essere usato solo su sub-espressioni booleane A<=!=>B puo essere tradotto in *A != B***	**booleano**	

Liskov

Gli oggetti della sotto-classe devono rispettare il contratto della super-classe.

In altre parole il comportamento della sotto-classe deve essere compatibile con il comportamento della super-

classe.
Un oggetto di un tipo deve essere in grado di usare un oggetto di un tipo derivato senza "notare" le differenza.

Per vedere se il una sotto-classe rispetta Liskov si possono fare le seguenti domande:

1. Cambiano i metodi della classe?
2. Si potrebbe fare la stessa cosa con i metodi già esistenti?
3. Restano invariate le proprietà?

In termini JML-ani bisogna assicurarsi che la @Requires sia uguale o piu permissiva (piu corta) e l'@Ensures sia uguale o piu stringente (piu lunga).

Inheritance and Visibility

```
/*
public class Utils
{
    public static void print(Object... mexs)
    {
        for(Object mex : mexs) System.out.println(mex);
    }
}
*/

import static com.mycompany.inheritancetest.Utils.*;


class A
{
    public String name;

    A(String name)
    {
        this.name = name;
    }

    public void shout()
    {
        print("Class A - Print from " + name);
    }

    public void shoutVisibleA()
    {
        print("Class A - Print visible from " + name);
```

```
    }

    private void shoutHiddenA()
    {
        print("Class A - Print hidden from " + name);
    }
}

class B extends A
{
    public String name; //questo nascode il nome di A

    B(String name)
    {
        super(name);  // chiama il costruttore delle classe A
        this.name = name;
    }

    @Override
    public void shout()
    {
        print("Class B - Print from " + name);
    }

    public void shoutVisibleB()
    {
        print("Class B - Print visible from " + name);
    }
}

class C extends B
{
    public String name;  //questo nasconde il nome di B

    C(String name)
    {
        super(name);  // chiama il costruttore delle classe B
        this.name = name;
    }

    @Override
    public void shout()
    {
```

```
        print("Class C - Print from " + name);
    }

    public void shoutVisibleC()
    {
        print("Class C - Print visible from " + name);
    }
}


public class InheritanceTest {

    public static void main(String[] args) {
        A a1 = new A("a1");
        A a2 = new B("a2");
        A a3 = new C("a3");

        B b1 = new B("b1");
        B b2 = new C("b2");
        //B b3 = new A("b3"); /*illegal: A cannot be converted to B*/

        C c1 = new C("c1");
        //C c2 = new B("c2"); /*illegal: B cannot be converted to C*/
        //C c3 = new A("c3"); /*illegal: A cannot be converted to C*/

        //---------------

        a1.shout();
        a1.shoutVisibleA();
        //a1.shoutHiddenA(); /*illegal: shoutHiddenA è privato quindi non
puo essere chiamato*/
        //----------------
        a2.shout();
        a2.shoutVisibleA();
        //a2.shoutVisibleB(); /*illegal: cannot find symbol shoutVisibleB()
in a2*/
        //----------------
        a3.shout();
        a3.shoutVisibleA();
        //a3.shoutVisibleB(); /*illegal: cannot find symbol shoutVisibleB()
in a3*/
        //a3.shoutVisibleC(); /*illegal: cannot find symbol shoutVisibleC()
in a3*/
```

```
        //----------------

        b1.shout();
        b1.shoutVisibleB();
        b1.shoutVisibleA();
        //----------------
        b2.shout();
        b2.shoutVisibleB();
        b2.shoutVisibleA();
        //b2.shoutVisibleC(); /*illegal: cannot find symbol shoutVisibleC()
in b2*/

        //----------------

        c1.shout();
        c1.shoutVisibleC();
        c1.shoutVisibleB();
        c1.shoutVisibleA();
    }
}


/*
OUTPUT::


Class A - Print from a1
Class A - Print visible from a1

Class B - Print from a2
Class A - Print visible from a2

Class C - Print from a3
Class A - Print visible from a3

Class B - Print from b1
Class B - Print visible from b1
Class A - Print visible from b1

Class C - Print from b2
Class B - Print visible from b2
Class A - Print visible from b2
```

```
Class C - Print from c1
Class C - Print visible from c1
Class B - Print visible from c1
Class A - Print visible from c1


*/
```

Testing

Statement Coverage

Deve entrare una volta in tutti gli if, else, while,for, ecc...

Edge Coverage

Come lo statement covarage, ma deve entrare anche nelle istruzioni "sottointese", per esempio, in un if senza else, deve entrare anche nell'else "mancante"

Path Coverage

Come l'edge coverage ma deve percorrere tutti i percorsi possibili, quindi tutte le combinazioni possibili di if e else, se ci sono loop sono da considerare anche il numero d'ingressi nell'loop se con valori diversi entra un numero diverso di volte è da considerare come un altro percorso.