Easter Exercise

Antonio Petrillo

April 12, 2023

Contents

1 Esercizio 1 **1**
1.1 Note 4

2 Esercizio 2 **4**
2.1 Contratto a 4
2.1.1 Risposta 5
2.2 Contratto b 5
2.2.1 Risposta 5
2.3 Contratto c 5
2.3.1 Risposta 5
2.4 Contratto d 5
2.4.1 Risposta 5

3 Esercizio 3 **6**
3.1 Memory layout 6
3.2 Graficamente 8

4 Esercizio 4 **8**

1 Esercizio 1

Implementare WeightedSet.

```
import java.util.Iterator;
import java.util.SortedSet;
import java.util.TreeSet;

public class MyWeightedSet<T> {
```

```
private static class Entry<E> implements Comparable<Entry<?>> {
    E element;
    Integer weight;

    Entry(E element, Integer weight) {
        this.element = element;
        this.weight = weight;
    }

    @Override
    public int compareTo(Entry<?> other) {
        return Integer.compare(weight, other.weight);
    }

    @Override
    public String toString() {
        return element.toString();
    }
}

private SortedSet<Entry<T>> set;
private Integer threshold;

public MyWeightedSet() {
    this.set = new TreeSet<>();
    this.threshold = null;
}

private boolean checkThreshold(Integer weight) {
    return threshold == null || threshold != null && weight >= threshold;
}

public boolean add(T t, Integer weight) {
    Entry<T> e = new Entry<>(t, weight);
    if (checkThreshold(weight)) {
        if (set.contains(e)) {
            set.remove(e);
            // questa parte é un pó strana ma l'esempio del caso d'uso
            // fa vedere che l'Entry (new Object, 5) viene sostituita da ("Jesse",
```

```
            // usando il normale add di SortedSet sarei ("Jesse", 5) non verrebbe r
            // aggiunto
        }
        return set.add(e);
    }
    return false;
}

private MyWeightedSet(SortedSet<Entry<T>> set, Integer threshold) {
    this.set = set;
    this.threshold = threshold;
}

public MyWeightedSet<T> atLeast(Integer threshold) {
    return new MyWeightedSet<T>(set, threshold);
}

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("[");
    Iterator<? extends Entry<?>> iter = set.iterator();
    while (iter.hasNext()) {
        Entry<?> e = iter.next();
        if (checkThreshold(e.weight)) {
            sb.append(e);
            if (iter.hasNext()) {
                sb.append(", ");
            }
        }
    }
    sb.append("]");
    return sb.toString();
}

// sed -i -e '90s/ar/er/g' MyWeightedSet.java
public static void main(String[] args) {
    MyWeightedSet<Object> set = new MyWeightedSet<>();
    set.add(Double.valueOf(3.14), 100);
    set.add(new Object(), 5);
```

```
        set.add("Skylar", 50);
        set.add("Jesse", 5);
        System.out.println(set);
        MyWeightedSet<Object> set10 = set.atLeast(10);
        System.out.println(set10);
        set.add("Walter", 60);
        System.out.println(set);
        System.out.println(set10);
    }
}
```

1.1 Note

I motivi principali per cui `WeightedSet` non estende `Set` sono:

1. troppi metodi non richiesti da implementare
2. il metodo `add` di Set non é compatibile con il caso d'uso dell'esempio

Sempre nel metodo `add`, come si evince dal caso d'uso, nel momento in cui viene aggiunto un nuovo oggetto, chiamiamolo `A`, con un valore giá presente, chiamiamolo `B`, allora `A` dovrá sostituire `B`. Questo é il motivo per cui il metodo add fa ulteriori controlli. Ció puó essere visto nel caso d'uso, `set.add(new Object(), 5)` viene sostituito da `set.add("Jesse", 5)`.

2 Esercizio 2

Dato l'interfaccia:

```
interface Predicate<T> {
    boolean test(T t);
}
```

Dire quali dei seguenti contratti per un comparatore di `Predicate` sono validi.

2.1 Contratto a

-1 se `x.test(...)` sempre falso e `y.test(...)` sempre vero. 1 se `y.test(...)` sempre falso e `x.test(...)` sempre vero.

2.1.1 Risposta

Il contratto non é valido, non é transitivo. Supponiamo x, y, z tali che `compare(x,y) => -1` e `compare(y,z) => -1` dovremmo avere che `compare(x,z) => -1` ma questo non é garantito dalla specifica.

2.2 Contratto b

-1 se per tutti gli oggetti t il valore di `x.test(t)` é l'opposto di `y.test(t)`.
1 se per tutti gli oggetti t il valore di `x.test(t)` é uguale di `y.test(t)`.

2.2.1 Risposta

Il contratto non é valido, in particolare non é antisimmetrico. Consideriamo il seguente caso: Prendiamo `x, y` tale che `x.test(t) == ! y.test(t)` per un qualche `t`. In questo caso il contratto specifica che `x.test(t) == !y.test(t)` produce `-1`. Invertendo l'ordine di x e y si ha: `y.test(t) == !x.test(t)` che produce sempre `-1` e non `1`, questo perché se é vero che `x.test(t)` produce sempre un valore opposto a `y.test(t)` é anche vero il contrario.

2.3 Contratto c

-1 se l'insieme degli oggetti t per cui `x.test(t)` restituisce vero é un sottoinsieme proprio dell'insieme degli oggetti per cui `y.test(t)` restitituisce vero. 1 se esiste un oggetto t tale che `x.test(t)` restituisce vero e `y.test(t)` restituisce falso.

2.3.1 Risposta

Il contratto é valido

2.4 Contratto d

-1 se ci sono almeno 10 oggetti diversi su cui `x.test(t)` restituisce vero e `y.test(t)` restituisce falso. 1 se ci sono almeno 10 oggetti diversi su cui `x.test(t)` restituisce false e `y.test(t)` restituisce vero.

2.4.1 Risposta

Il contratto é non valido, in particolare non é transitivo. Supponiamo di avere 3 istanze x, y, z tali che ci sono: Almeno 10 elementi t_i tali che $x.test(t_i)$ sia

vero e $y.test(t_i)$ sia falso, inoltre ci sono almeno altri 10 elementi k_i tali che $y.test(k_i)$ sia vero e $z.test(k_i)$ sia falso. Per le specifiche di comparable ci aspetteremmo che `compare(x, z) == -1`, ma nessuno garantisce che esistano 10 elementi v_i per cui $x.test(v_i)$ sia vero e $z.test(v_i)$ sia falso.

3 Esercizio 3

Data la classe:

```
public class A {

    private A other;

    public A(A other) {
        this . other = other;
    }

public class B {
    private static int counter = 0;
    private int id = counter++;
}

    public Object makeObj(int val) {
        return new B() { private int j = val ;};
    }
}
```

Ed il seguente snippet:

```
A a1 = new A(null);
A a2 = new A(a1);
A.B b = a1.new B();
Object x = a1.makeObj(42);
A.B y = (A.B) a2.makeObj(42);
```

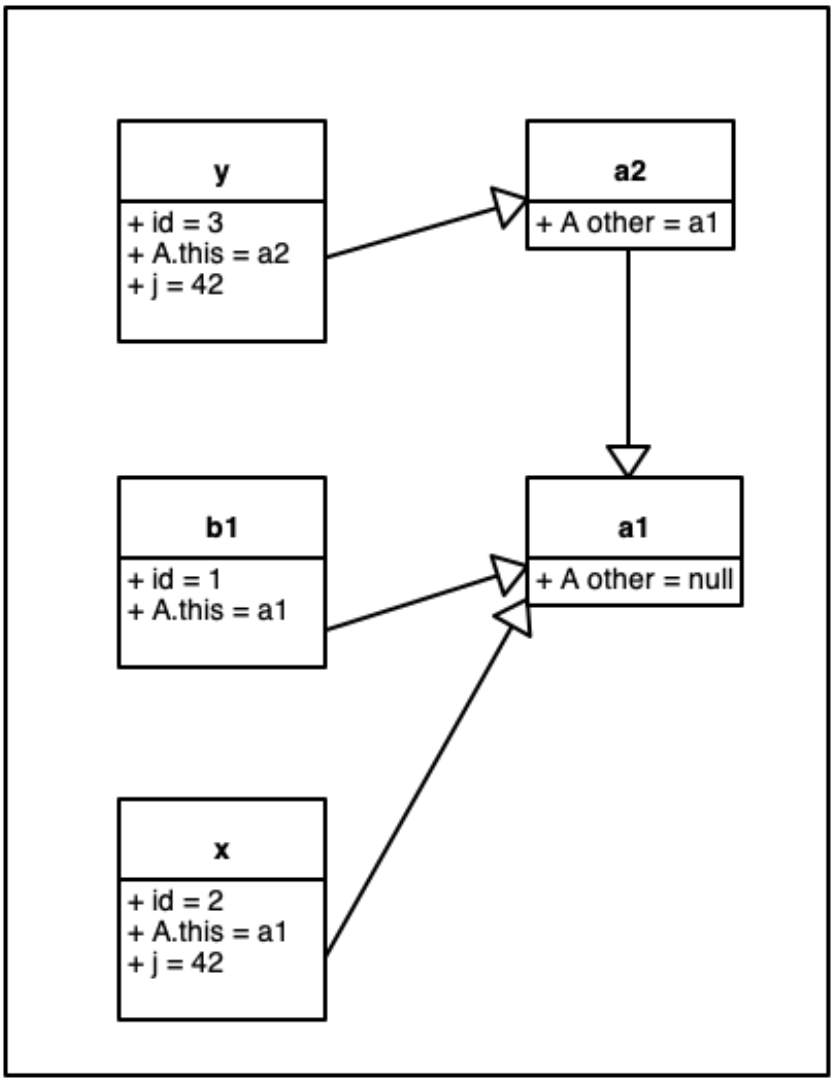
Disegnare il memory layout

3.1 Memory layout

- a1.other -> null
- a2.other -> a1

- b1.id = 1
- b1.A.this -> a1
- x.id = 2
- x.j = 42
- x.A.this -> a1
- y.id = 1
- y.j = 42
- y.A.this -> a2

3.2 Graficamente



4 Esercizio 4

Question	Answer
AR<Int> subtype of L<? ext Num>	True
Set<? ext Num> subtype of Set<? sup Num>	False
Map<Str, ? ext Num> subtype of Map<Object, ?>	False
TreeSet<Integer> subtype of SortedSet<? super Integer>	True
HashMap<Integer, Double> subtype of Map<?, ? super Double>	True