INSIEMI (SET)

CARATTERISTICHE GENERALI

- Contenitore di oggetti **distinti** (non duplicati)
- Corrisponde alla notazione matematica di **insieme**
- Non ha particolare ordinamento o memoria dell'ordine in cui gli oggetti sono inseriti/estratti
- Operazioni primitive
  - inserimento
  - verifica della presenza di un oggetto
  - ispezione di tutti gli oggetti
    - restituisce array
  - NO rimozione
- Operazioni insiemistiche
  - unione: $A \cup B$
  - intersezione: $A \cap B$
  - sottrazione: $A - B$

INTERFACCIA

INSIEME DATI NON ORDINATI

```
public interface Set extends Container {
    void add(Object obj);
    boolean contains(Object obj);
    Object[] toArray();
}
```

INSIEME DATI ORDINATI

```
public interface SortedSet implements Set {
    void add(Comparable obj);
    Comparable[] toSortedArray();
}
```

- Non sta sovrascrivendo **add** perché il type del parametro è diverso
  - nel metodo add originario dovrò lanciare una eccezione nell'implementazione

IMPLEMENTAZIONE IN JAVA DI ARRAYSET

- Metodo **contains**
  - controllo se l'oggetto esiste attraverso ricerca lineare

```
public boolean contains(Object x) {
    for (int i = 0; i<arraySize; i++) {
        if (array[i].equals(x)) return true;
    }
    return false;
}
```

- Metodo **add**
  - aggiunge l'oggetto nell'array dopo aver **verificato se non esiste già**
- Metodo **toArray**
  - restituisce nuovo array (con arraycopy) grande `arraySize`
- Metodo **union**

```
public static Set union(Set s1, Set s2) {
    Set mergedSet = new ArraySet();

    Object[] array1 = s1.toArray();
    Object[] array2 = s2.toArray();

    for (int i = 0; i<array1.length; i++) {
        mergedSet.add(array1[i]);
    }
    for (int i = 0; i<array2.length; i++) {
        mergedSet.add(array2[i]);
    }

    return mergedSet;
}
```

- Metodo **intersection**

```java
public static Set intersection(Set s1, Set s2) {
    Set mergedSet = new ArraySet();

    Object[] array1 = s1.toArray();

    for (int i = 0; i<array1.length; i++) {
        if (s2.contains(array1[i])) {
            mergedSet.add(array1[i]);
        }
    }

    return mergedSet;
}
```

- Metodo **subtract**

```java
public static Set subtract(Set s1, Set s2) {
    Set mergedSet = new ArraySet();

    Object[] array1 = s1.toArray();

    for (int i = 0; i<array1.length; i++) {
        if (!s2.contains(array1[i])) {
            mergedSet.add(array1[i]);
        }
    }

    return mergedSet;
}
```

- Operazioni primitive: $O(n)$
- Operazioni tra insiemi: $O(n^2)$

IMPLEMENTAZIONE IN JAVA DI ARRAY SORTED SET

- Mantengo array già ordinato
- Metodo **contains**
  - usa linear-search
  - prestazioni $O(\log n)$
- Metodo **add**
  - usa contains per verificare se oggetto è presente $O(\log n)$
  - deve usare **insertion sort** per mantenere array ordinato: $O(n)$
  - prestazioni complessive: $O(n) + O(\log n) = O(\log n)$

- Metodo **union**
  - sfruttare fusione di due array già ordinati (come merge sort)
  - viene eseguito `add` n volte, il quale internamente:
    - usa contains per verificare se oggetto è presente: $O(\log n)$
    - aggiunge elemento alla fine dell'array
    - poiché l'array parziale è già ordinato, l'algoritmo insertion sort è $O(1)$
  - prestazioni complessive: $O(n \log n)$

```
public static SortedSet union(SortedSet s1, SortedSet s2) {
    SortedSet newSet = new ArraySortedSet();
    Comparable[] array1 = s1.toSortedArray();
    Comparable[] array2 = s2.toSortedArray();

    int i=0, i1 = 0, i2 = 0;
    while (i1 < array1.length && i2 < array2.length) {
        if (array1[i1].compareTo(array2[i2]) < 0) {
            newSet.add(array1[i1++]);
        } else if (array2[i2].compareTo(array1[i1]) < 0) {
            newSet.add(array2[i2++]);
        } else {
            newSet.add(array1[i1]);
            i1++;
            i2++;
        }
    }

    return newSet;
}
```

ANALISI PRESTAZIONI

Prestazioni in sintesi

- add: $O(n)$
- toArray: $O(n)$
- toSortedArray: $O(n)$
- contains: $O(n)$ (non ordinato) - $O(\log n)$ (ordinato)
- union: $O(n^2)$ (non ordinato) - $O(n \log n)$ (ordinato)
- intersection: $O(n^2)$ (non ordinato) - $O(n \log n)$ (ordinato)
- subtract: $O(n^2)$ (non ordinato) - $O(n \log n)$ (ordinato)