

Estructura de Dades i Algorismes (EDA)

Grau en Enginyeria Informàtica de Gestió i
Sistemes d'Informació (**GEISI**)

Doble Grau en Informàtica de Gestió i
Sistemes d'Informació/ Grau en Disseny i
Producció de Videojocs (**DB GEISI-GDPV**)

Centre universitari adscrit a la



Universitat
Pompeu Fabra
Barcelona



Col·leccions i magatzems de dades [1]

Diferents maneres d'emmagatzemar objectes



 **Tecnocampus**

Centre universitari adscrit a la



**Universitat
Pompeu Fabra
Barcelona**

Una **col·lecció de dades** (*col·lection*)
és una classe els objectes de la qual permeten
l'emmagatzemament de dades (objectes). En poques
paraules **una col·lecció de dades és un magatzem**

Parlarem indistintament de *col·lecció de dades* o de
col·lecció d'objectes

Una **estructura de dades** és una *infraestructura* que
possibilita la implementació de **col·leccions de dades**.

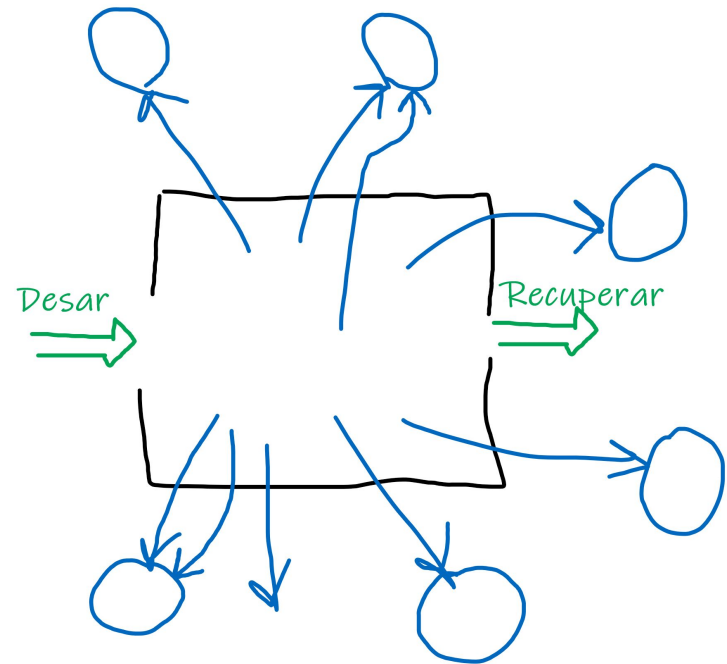
A
B
A
N
S

```
Ordinador o1;  
Ordinador o2;  
o1 = new Ordinador(...);  
o2 = new Ordinador(...);  
...
```

```
Ordinador [] taulaOrdinador;  
taulaOrdinador = new taulaOrdinador[20];  
...  
taulaOrdinador[0]=new Ordinador(...);  
taulaOrdinador[1]=new Ordinador(...);  
...  
TaulaOrdinador[19]=new Ordinador(...);
```

*Cal més versatilitat!!!
sobretot si treballem amb mooolts objectes*

A
R
A



El terme **col·lecció de dades** fa referència a:

- Quin **comportament** ha de tenir el magatzem
- Quines **operacions** ha de proporcionar el magatzem

El **comportament** i les **operacions** estan relacionats:

- 1) quines operacions ha de proporcionar i
- 2) quin comportament han de tenir

En *Java*, una **col·lecció de dades** (**collection**) es defineix, en primera instància, com una **Interfície**.

És a dir, s'indica quines són les operacions i quin és el comportament que es desitja, però no s'indica com implementar-lo.

Mentre que el terme **estructura de dades** fa referència a **com es construeixen** els magatzems.

MAGATZEM

estructura
de dades

Comportament
ve definit per
OPERACIONS

Desar

proporcionar
qualcom desat
(recuperar)

- Organització interna
del magatzem -

En funció del comportament esperat, existeixen diferents menes de col·leccions:

Piles

stacks

Cues

queues

Llistes

Lists

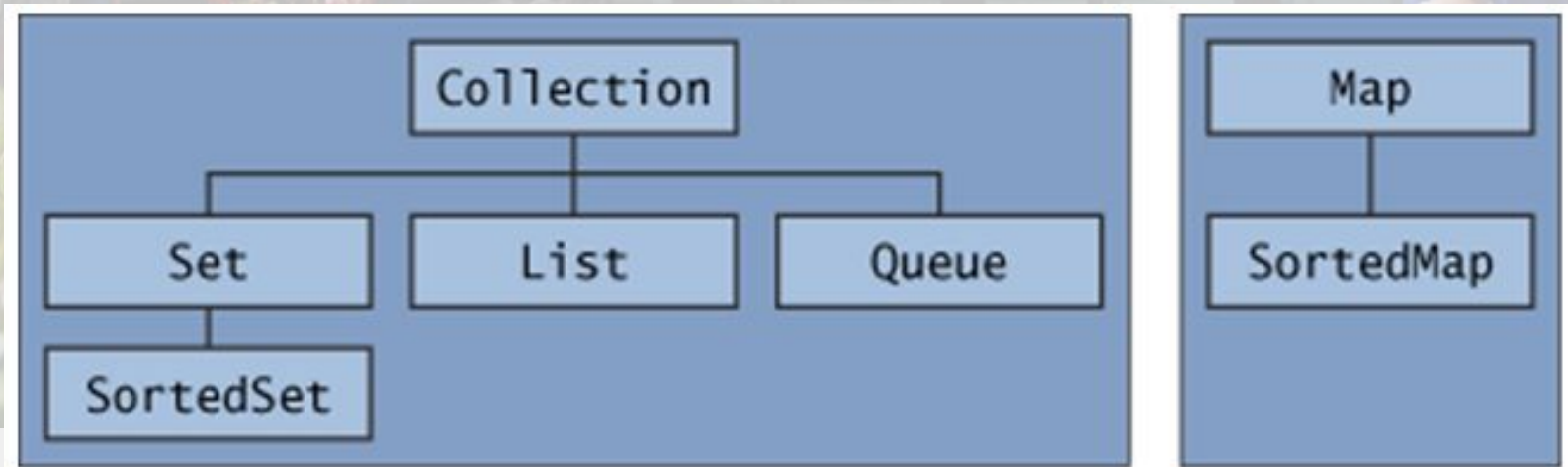
Conjunts i bosses

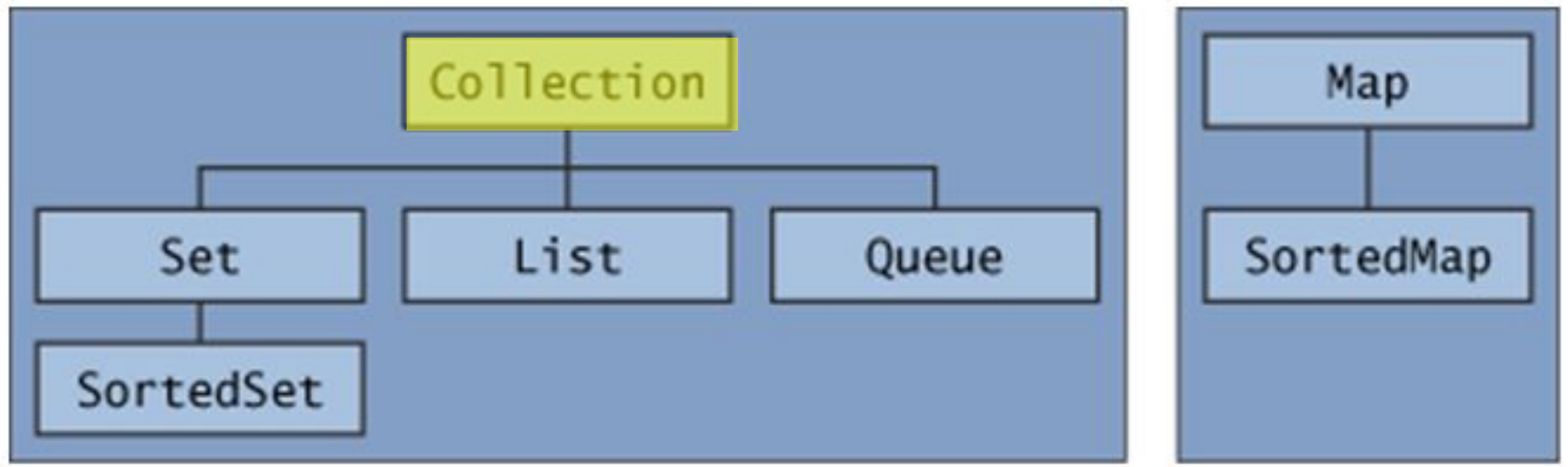
sets i *bags*

Diccionaris

maps o *funcions*

- Java a través del paquet standard ***java.util*** proporciona un ampli ventall de magatzems de dades amb diferents implementacions.
- S'anomena ***Java Collection Framework (JCF)*** al conjunt de les interfícies que defineixen magatzems i les classes que les implementen.
- Les interfícies més rellevants del ***JCF*** són:





- L'organització de les **interfícies** és **jeràrquica**
- **Map** i **SortedMap** també defineixen magatzems, però el **JCF** no els considera col·leccions.

El que realment és important, és que són **magatzems**.

Collection defineix diferents grups de mètodes

- Mètodes per **afegir contingut** a la col·lecció: *add* i *addAll*
- Mètodes per **determinar la presència d'objectes** a la col·lecció: *contains* i *containsAll*
- Mètodes per **eliminar objectes** de la col·lecció: *remove*, *removeAll*, *retainAll* i *clear*
- Mètodes per **passar el contingut a una taula**: *toArray*
- Mètode per **obtenir un repetidor**: *iterator*
- Altres mètodes: *isEmpty*, *size*...






Fitxer amb la documentació [java1.4.2.zip](#)

Contingut basat en 1.4.2

[Collection from Java SE 22 & JDK 22 docs.oracle.com](#)

Method Summary

[Interface Collection<E> Java SE 22 & JDK 22](#)

boolean 	<u>add</u> (<u>Object</u> o) Ensures that this collection contains the specified element (optional operation).
boolean 	<u>addAll</u> (<u>Collection</u> c) Adds all of the elements in the specified collection to this collection (optional operation).
void 	<u>clear</u> () Removes all of the elements from this collection (optional operation).
boolean 	<u>contains</u> (<u>Object</u> o) Returns true if this collection contains the specified element.
boolean 	<u>containsAll</u> (<u>Collection</u> c) Returns true if this collection contains all of the elements in the specified collection.
boolean	<u>equals</u> (<u>Object</u> o) Compares the specified object with this collection for equality.
int	<u>hashCode</u> () Returns the hash code value for this collection.

boolean ●	<u>isEmpty()</u> Returns true if this collection contains no elements.
<u>Iterator</u> ●	<u>iterator()</u> Returns an iterator over the elements in this collection.
boolean ●	<u>remove()</u> (<u>Object</u> o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean ●	<u>removeAll()</u> (<u>Collection</u> c) Removes all this collection's elements that are also contained in the specified collection (optional operation).
boolean ●	<u>retainAll()</u> (<u>Collection</u> c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
int ●	<u>size()</u> Returns the number of elements in this collection.
<u>Object</u> [] ●	<u>toArray()</u> Returns an array containing all of the elements in this collection.
<u>Object</u> []	<u>toArray()</u> (<u>Object</u> [] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

Class Vector

[Vector<E> Java SE 22 & JDK 22](#)

[java.lang.Object](#)
└ [java.util.AbstractCollection](#)
 └ [java.util.AbstractList](#)
 └ [java.util.Vector](#)

All Implemented Interfaces:

[Cloneable](#), [Collection](#), [List](#), [RandomAccess](#), [Serializable](#)

Direct Known Subclasses:

[Stack](#)



```
public class Vector  
extends AbstractList  
implements List, RandomAccess, Cloneable, Serializable
```

The `Vector` class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a `Vector` can grow or shrink as needed to accommodate adding and removing items after the `Vector` has been created.

Class LinkedList

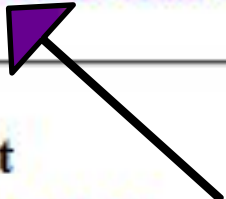
[LinkedList<E> Java SE 22 & JDK 22](#)

```
java.lang.Object
├── java.util.AbstractCollection
│   ├── java.util.AbstractList
│   │   ├── java.util.AbstractSequentialList
│   │   └── java.util.LinkedList
```

All Implemented Interfaces:

[Cloneable](#), [Collection](#), [List](#), [Serializable](#)

public class **LinkedList**
extends [AbstractSequentialList](#)
implements [List](#), [Cloneable](#), [Serializable](#)



Linked list implementation of the `List` interface. Implements all optional list operations, and permits all elements (including `null`). In addition to implementing the `List` interface, the `LinkedList` class provides uniformly named methods to get, remove and insert an element at the beginning and end of the list. These operations allow linked lists to be used as a stack, queue, or double-ended queue (deque).

Class ArrayList

[ArrayList<E> Java SE 22 & JDK 22](#)[java.lang.Object](#)└ [java.util.AbstractCollection](#)└ [java.util.AbstractList](#)└ [java.util.ArrayList](#)

All Implemented Interfaces:

[Cloneable](#), [Collection](#), [List](#), [RandomAccess](#), [Serializable](#)

```
public class ArrayList
```

```
extends AbstractList
```

```
implements List, RandomAccess, Cloneable, Serializable
```

Resizable-array implementation of the `List` interface. Implements all optional list operations, and permits all elements, including `null`. In addition to implementing the `List` interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to `Vector`, except that it is unsynchronized.)


```
public class Element {
```

```
    private String name;
    private int value;
```

```
// Constructor
```

```
public Element (String name, int value) {
    this.name = name;
    this.value = value;
}
```

```
// setters & getter
```

```
public String getName() {return this.name;}
public int getValue() {return this.value;}
public void setValue(int newVal) {this.value=newVal;}
```

```
// equality (overriding of superclass equals)
```

```
public boolean equals (Object o) {
    Element other;
    try {
        other = (Element)o;
        return this.name.equals(other.name);
    }
    catch(Exception e) {
        return false;
    }
}
```

```
// toString (overriding of superclass toString)
```

```
public String toString () {
    return "Element["+name+" "+value+"]";
}
```

```
}
```

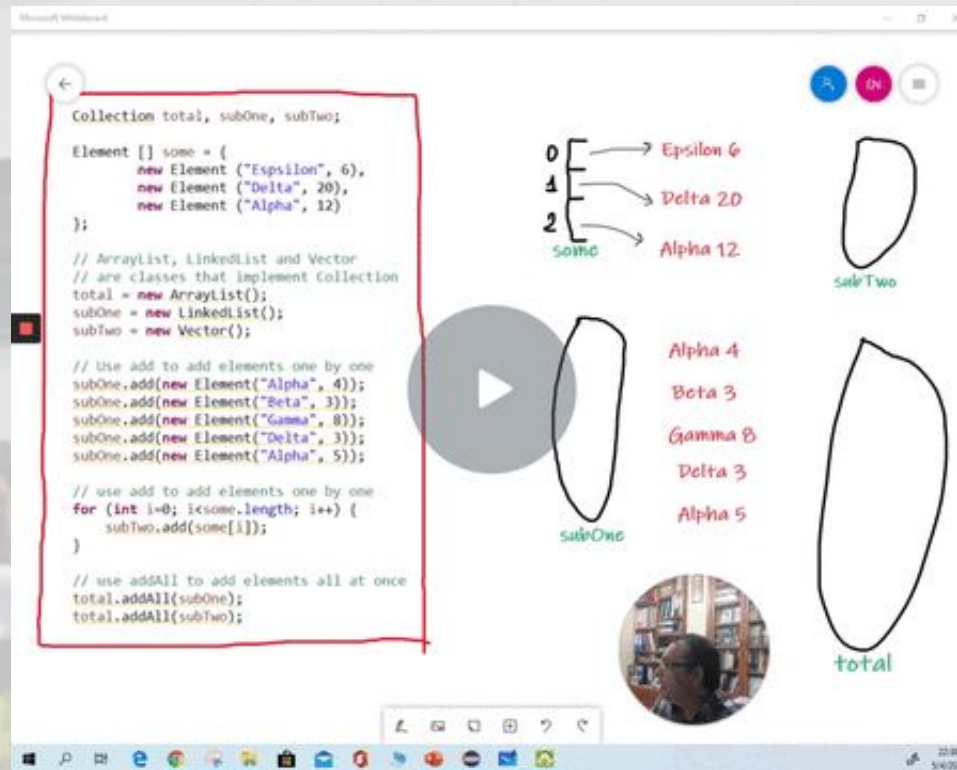
Exemples

Els objectes de la classe **Element** seran **guardats en diferents magatzems Collections**



Observem que s'ha **implementat redefinit equals**, perquè dos elements siguin considerats iguals **si tenen el mateix nom**

- Enllaç al vídeo -



The screenshot shows a video player interface. On the left, a code editor displays Java code for managing a collection of elements. On the right, three hand-drawn diagrams illustrate the state of the collections: 'Some', 'subOne', and 'total'. A large play button is centered over the diagrams.

```
Collection total, subOne, subTwo;

Element [] some = {
    new Element ("Epsilon", 6),
    new Element ("Delta", 20),
    new Element ("Alpha", 12)
};

// ArrayList, LinkedList and Vector
// are classes that implement Collection
total = new ArrayList();
subOne = new LinkedList();
subTwo = new Vector();

// Use add to add elements one by one
subOne.add(new Element("Alpha", 4));
subOne.add(new Element("Beta", 3));
subOne.add(new Element("Gamma", 8));
subOne.add(new Element("Delta", 3));
subOne.add(new Element("Alpha", 5));

// use add to add elements one by one
for (int i=0; i<some.length; i++) {
    subTwo.add(some[i]);
}

// use addAll to add elements all at once
total.addAll(subOne);
total.addAll(subTwo);
```

Diagram 1: Some

- 0 → Epsilon 6
- 1 → Delta 20
- 2 → Alpha 12

Diagram 2: subOne

- Alpha 4
- Beta 3
- Gamma 8
- Delta 3
- Alpha 5

Diagram 3: total

The 'total' diagram is an empty container, representing the result of adding 'subOne' and 'subTwo' to it.

```
Collection total, subOne, subTwo;
```

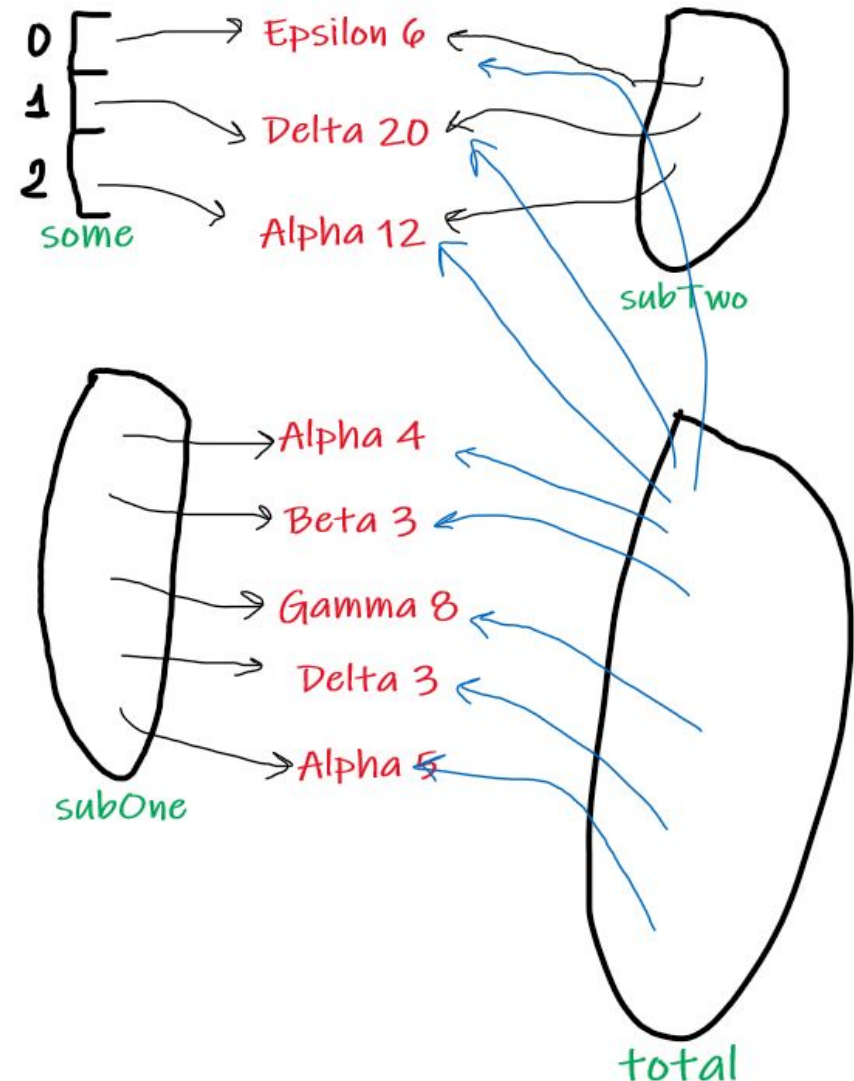
```
Element [] some = {  
    new Element ("Epsilon", 6),  
    new Element ("Delta", 20),  
    new Element ("Alpha", 12)  
};
```

```
// ArrayList, LinkedList and Vector  
// are classes that implement Collection  
total = new ArrayList();  
subOne = new LinkedList();  
subTwo = new Vector();
```

```
// Use add to add elements one by one  
subOne.add(new Element("Alpha", 4));  
subOne.add(new Element("Beta", 3));  
subOne.add(new Element("Gamma", 8));  
subOne.add(new Element("Delta", 3));  
subOne.add(new Element("Alpha", 5));
```

```
// use add to add elements one by one  
for (int i=0; i<some.length; i++) {  
    subTwo.add(some[i]);  
}
```

```
// use addAll to add elements all at once  
total.addAll(subOne);  
total.addAll(subTwo);
```



Snapshots del video

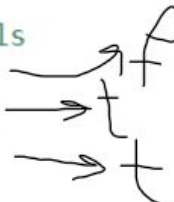
2/4

```
// contains and containsAll are based on equals
```

```
System.out.println(subOne.contains(some[0]));
```

```
System.out.println(subOne.contains(some[1]));
```

```
System.out.println(subOne.contains(some[2]));
```



```
// remove and removeAll are based on equals
```

```
subOne.remove(some[1]);
```

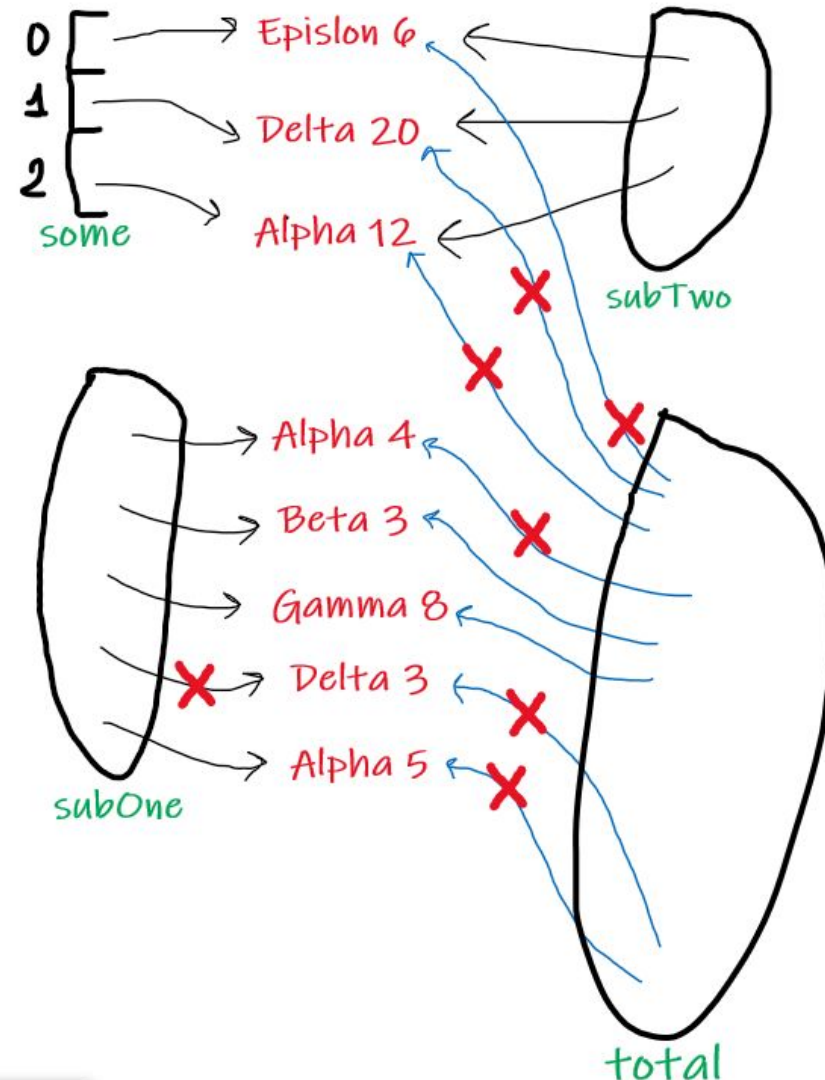
```
total.removeAll(subTwo);
```

boolean	<code>contains(Object o)</code>
●	Returns true if this collection contains the specified element.

boolean	<code>remove(Object o)</code>
●	Removes a single instance of the specified element from this collection, if it is present (optional operation).

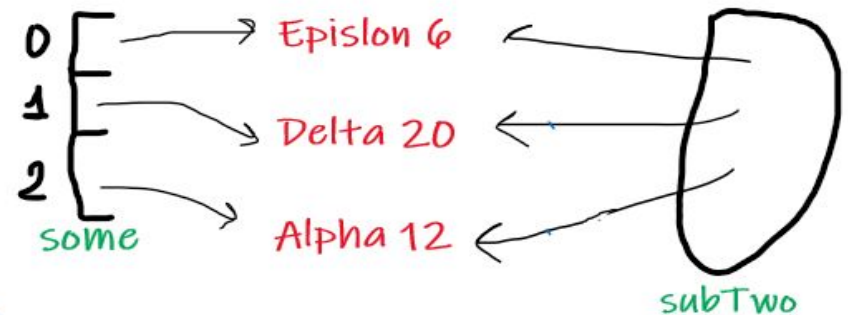
boolean	<code>removeAll(Collection c)</code>
●	Removes all this collection's elements that are also contained in the specified collection (optional operation).

EQUALS

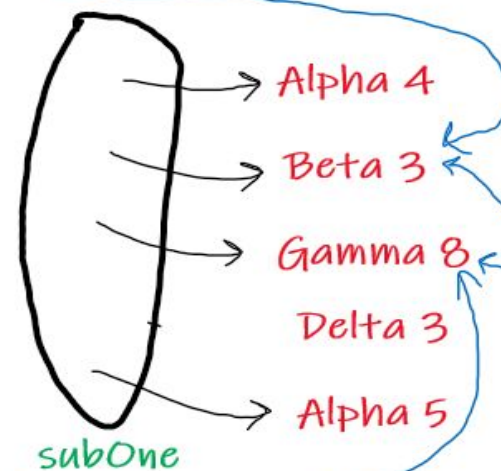


```
// toArray provides the contents in an array
Object [] contents = total.toArray();

for (int i=0; i<contents.length; i++) {
    System.out.println(contents[i]);
}
```



Element[Beta 3]
Element[Gamma 8]

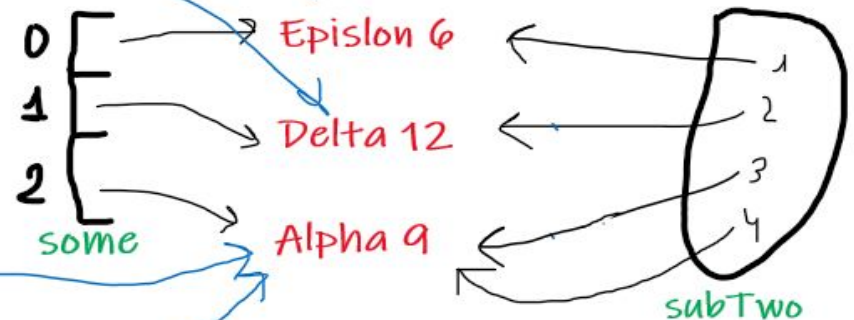


Object[] toArray()



Returns an array containing all of the elements in this collection.

```
// Remember: REFERENCES EVERYWHERE
some[1].setValue(12);
some[2].setValue(some[0].getValue()+3);
// Remember: REFERENCES EVERYWHERE
subTwo.add(some[2]);
contents = subTwo.toArray();
int sum = 0;
for (int i=0; i<contents.length; i++) {
    System.out.println(contents[i]);
    sum = sum + ((Element)contents[i]).getValue();
}
System.out.println("Sum is: "+sum);
```



Element[Epsilon 6]
Element[Delta 12]
Element[Alpha 9]
Element[Alpha 9]
Sum is: 36