Col·leccions: Més enllà de les taules

PROP 2012-13 Q2

Mario Martin

Col·leccions i Maps

- El llenguatge Java proporciona implementacions per moltes estructures de dades
- Es defineix una "col·lecció" com "*un objecte que representa un grup d'objectes*".
- És "una arquitectura unificada per representar i manipular col·leccions, el que els permet ser manipulat independentment dels detalls de la seva representació".
- Totes a Java.util:

```
import java.util.*;
```

o

```
import java.util.Collection;
```

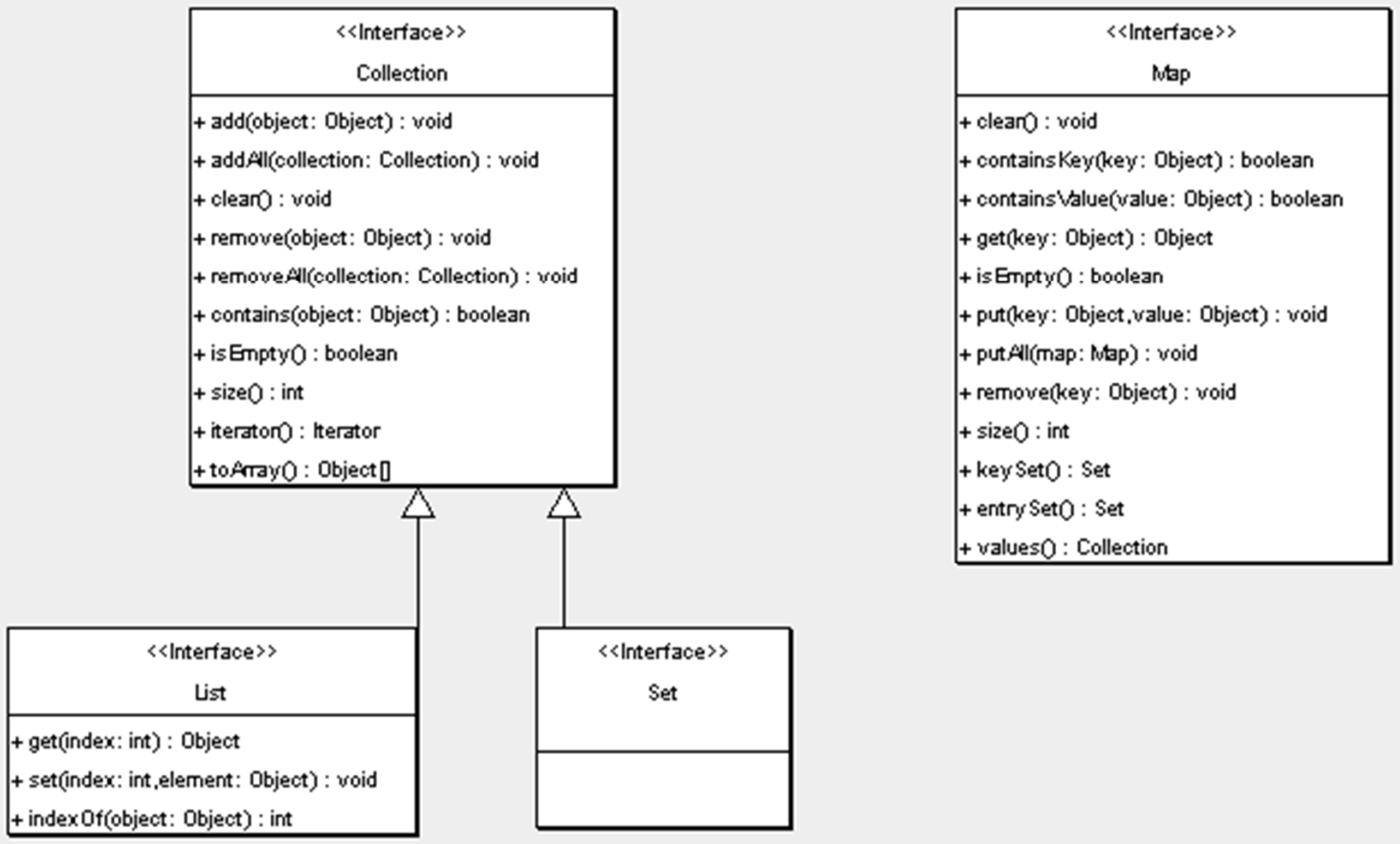
[Classes Genèriques]

- Les col·leccions són classes genèriques, és a dir, col·leccions de objectes de qualsevol classe.
- Exemples:
  - `ArrayList<Node>`
  - `LinkedList<String>`

[Classes Genèriques]

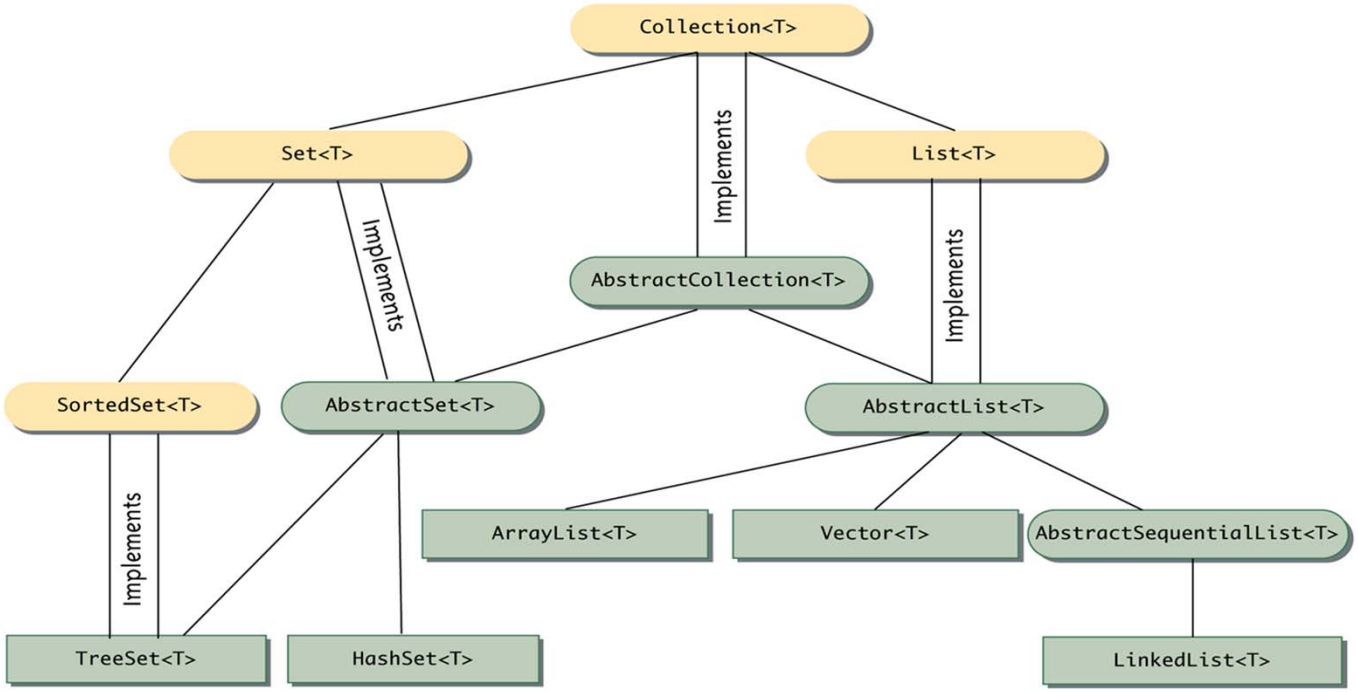
```
public class GenericClass<T> {
    private T obj;
    public void setObj(T t) {obj = t;}
    public T getObj() {return obj;}
    public void print() {
        System.out.println(obj);
    }
}
```

Collection<T>: Mètodes de la interfície



Collections

Display 16.1 The Collection Landscape



A single line between two boxes means the lower class or interface is derived from (`extends`) the higher one.

T is a type parameter for the type of the elements stored in the collection.

Col·leccions

Mètodes de la interfície

Collection<T>: Mètodes de la interfície

Display 16.2 Method Headings in the `Collection<T>` Interface

The `Collection<T>` interface is in the `java.util` package.
All the exception classes mentioned are unchecked exceptions, which means they are not required to be caught in a `catch` block or declared in a `throws` clause.
All the exception classes mentioned are in the package `java.lang` and so do not require any import statement.

CONSTRUCTORS

Although not officially required by the interface, any class that implements the `Collection<T>` interface should have at least two constructors: a no-argument constructor that creates an empty `Collection<T>` object, and a constructor with one parameter of type `Collection<? extends T>` that creates a `Collection<T>` object with the same elements as the constructor argument. The interface does not specify whether the copy produced by the one-argument constructor is a shallow copy or a deep copy of its argument.

```
boolean isEmpty()
```

Returns `true` if the calling object is empty; otherwise returns `false`.

(continued)

Collection<T>: Mètodes de la interfície

Display 16.2 Method Headings in the `Collection<T>` Interface

```
public boolean contains(Object target)
```

Returns `true` if the calling object contains at least one instance of `target`. Uses `target.equals` to determine if `target` is in the calling object.
Throws a `ClassCastException` if the type of `target` is incompatible with the calling object (optional).
Throws a `NullPointerException` if `target` is `null` and the calling object does not support `null` elements (optional).

(continued)

Collection<T>: Mètodes de la interfície

Display 16.2 Method Headings in the `Collection<T>` Interface

```
public boolean containsAll(Collection<?> collectionOfTargets)
```

Returns `true` if the calling object contains all of the elements in `collectionOfTargets`. For an `element` in `collectionOfTargets`, this method uses `element.equals` to determine if `element` is in the calling object.
Throws a `ClassCastException` if the types of one or more elements in `collectionOfTargets` are incompatible with the calling object (optional).
Throws a `NullPointerException` if `collectionOfTargets` contains one or more `null` elements and the calling object does not support `null` elements (optional).
Throws a `NullPointerException` if `collectionOfTargets` is `null`.

```
public boolean equals(Object other)
```

This is the `equals` of the collection, not the `equals` of the elements in the collection. Overrides the inherited method `equals`. Although there are no official constraints on `equals` for a collection, it should be defined as we have described in Chapter 7 and also to satisfy the intuitive notion of collections being equal.

(continued)

Collection<T>: Mètodes de la interfície

Display 16.2 Method Headings in the `Collection<T>` Interface

```
public int size()
```

Returns the number of elements in the calling object. If the calling object contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.

```
Iterator<T> iterator()
```

Returns an iterator for the calling object. (Iterators are discussed in Section 16.2.)

```
public Object[] toArray()
```

Returns an array containing all of the elements in the calling object. If the calling object makes any guarantees as to what order its elements are returned by its iterator, this method must return the elements in the same order.
The array returned should be a new array so that the calling object has no references to the returned array. (You might also want the elements in the array to be clones of the elements in the collection. However, this is apparently not required by the interface, since library classes, such as `Vector<T>`, return arrays that contain references to the elements in the collection.)

(continued)

Collection<T>: Mètodes de la interfície

Display 16.2 **Method Headings in the `Collection<T>` Interface**

```
public boolean add(T element) (Optional)
```

Ensures that the calling object contains the specified `element`. Returns `true` if the calling object changed as a result of the call. Returns `false` if the calling object does not permit duplicates and already contains `element`; also returns `false` if the calling object does not change for any other reason.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the class of `element` prevents it from being added to the calling object.
Throws a `NullPointerException` if `element` is `null` and the calling object does not support `null` elements.
Throws an `IllegalArgumentException` if some other aspect of `element` prevents it from being added to the calling object.

(continued)

Collection<T>: Mètodes de la interfície

Display 16.2 **Method Headings in the `Collection<T>` Interface**

```
public boolean addAll(Collection<? extends T> collectionToAdd) (Optional)
```

Ensures that the calling object contains all the elements in `collectionToAdd`. Returns `true` if the calling object changed as a result of the call; returns `false` otherwise. If the calling object changes during this operation, its behavior is unspecified; in particular, its behavior is unspecified if `collectionToAdd` is the calling object.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the class of an element of `collectionToAdd` prevents it from being added to the calling object.
Throws a `NullPointerException` if `collectionToAdd` contains one or more `null` elements and the calling object does not support `null` elements, or if `collectionToAdd` is `null`.
Throws an `IllegalArgumentException` if some aspect of an element of `collectionToAdd` prevents it from being added to the calling object.

(continued)

Collection<T>: Mètodes de la interfície

Display 16.2 Method Headings in the `Collection<T>` Interface

```
public boolean remove(Object element) (Optional)
```

Removes a single instance of the `element` from the calling object, if it is present. Returns `true` if the calling object contained the `element`; returns `false` otherwise.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the type of `element` is incompatible with the calling object (optional).
Throws a `NullPointerException` if `element` is `null` and the calling object does not support `null` elements (optional).

```
public boolean removeAll(Collection<?> collectionToRemove) (Optional)
```

Removes all the calling object's elements that are also contained in `collectionToRemove`. Returns `true` if the calling object was changed; otherwise returns `false`.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the types of one or more elements in `collectionToRemove` are incompatible with the calling collection (optional).
Throws a `NullPointerException` if `collectionToRemove` contains one or more `null` elements and the calling object does not support `null` elements (optional).
Throws a `NullPointerException` if `collectionToRemove` is `null`.

(continued)

Collection<T>: Mètodes de la interfície

Display 16.2 Method Headings in the `Collection<T>` Interface

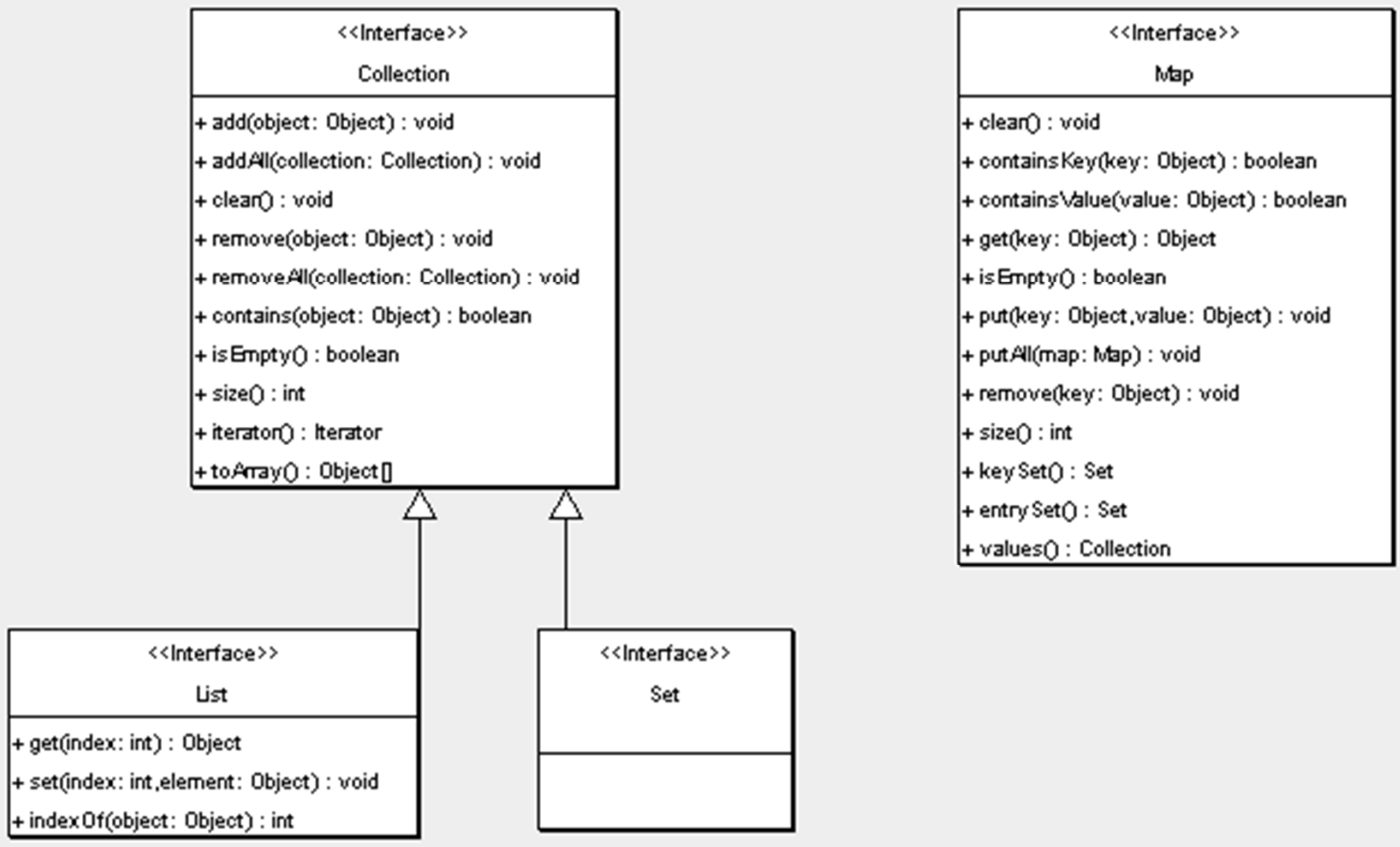
```
public void clear() (Optional)
```

Removes all the elements from the calling object.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.

```
public boolean retainAll(Collection<?> saveElements) (Optional)
```

Retains only the elements in the calling object that are also contained in the collection `saveElements`. In other words, removes from the calling object all of its elements that are not contained in the collection `saveElements`. Returns `true` if the calling object was changed; otherwise returns `false`.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the types of one or more elements in `saveElements` are incompatible with the calling object (optional).
Throws a `NullPointerException` if `saveElements` contains one or more `null` elements and the calling object does not support `null` elements (optional).
Throws a `NullPointerException` if `saveElements` is `null`.

Collection<T>: Mètodes de la interfície



Conjunts

Una implementació de les col·leccions

Conjunts: Set<T>

- Les classes que implementen la interfície **`Set<T>`** no permeten que un element de la classe aparegui més d'una vegada
- Els elements es comparen fent servir el métode `equals`
- [Compte amb afegir en un conjunt objectes mutables (poden canviar). Si canvien es poden violar les condicions de conjunt]

Set<T>: Mètodes de la interfície

Display 16.3 Methods in the `Set<T>` Interface

The `Set<T>` interface is in the `java.util` package.
The `Set<T>` interface extends the `Collection<T>` interface.
All the exception classes mentioned are the kind that are not required to be caught in a `catch` block or declared in a `throws` clause.
All the exception classes mentioned are in the package `java.lang` and so do not require any import statement.

CONSTRUCTORS

Although not officially required by the interface, any class that implements the `Set<T>` interface should have at least two constructors: a no-argument constructor that creates an empty `Set<T>` object, and a constructor with one parameter of type `Collection<? extends T>` that creates a `Set<T>` object with the same elements as the constructor argument.

(continued)

Set<T>: Mètodes de la interfície

Display 16.3 Methods in the `Set<T>` Interface

`boolean isEmpty()`

Returns `true` if the calling object is empty; otherwise returns `false`.

`public boolean contains(Object target)`

Returns `true` if the calling object contains at least one instance of `target`. Uses `target.equals` to determine if `target` is in the calling object.
Throws a `ClassCastException` if the type of `target` is incompatible with the calling object (optional).
Throws a `NullPointerException` if `target` is `null` and the calling object does not support `null` elements (optional).

(continued)

Set<T>: Mètodes de la interfície

Display 16.3 Methods in the Set<T> Interface

```
public boolean containsAll(Collection<?> collectionOfTargets)
```

Returns `true` if the calling object contains all of the elements in `collectionOfTargets`. For an `element` in `collectionOfTargets`, this method uses `element.equals` to determine if `element` is in the calling object. If `collectionOfTargets` is itself a `Set<T>`, this is a test to see if `collectionOfTargets` is a subset of the calling object.
Throws a `ClassCastException` if the types of one or more elements in `collectionOfTargets` are incompatible with the calling object (optional).
Throws a `NullPointerException` if `collectionOfTargets` contains one or more `null` elements and the calling object does not support `null` elements (optional).
Throws a `NullPointerException` if `collectionOfTargets` is `null`.

```
public boolean equals(Object other)
```

If the argument is a `Set<T>`, returns `true` if the calling object and the argument contain exactly the same elements; otherwise returns `false`. If the argument is not a `Set<T>`, `false` is returned.

(continued)

Set<T>: Mètodes de la interfície

Display 16.3 Methods in the Set<T> Interface

```
public int size()
```

Returns the number of elements in the calling object. If the calling object contains more than Integer.MAX_VALUE elements, returns Integer.MAX_VALUE.

```
Iterator<T> iterator()
```

Returns an iterator for the calling object. (Iterators are discussed in Section 16.2.)

```
public Object[] toArray()
```

Returns an array containing all of the elements in the calling object. A new array must be returned so that the calling object has no references to the returned array.

(continued)

Set<T>: Mètodes de la interfície

Display 16.3 Methods in the Set<T> Interface

ADDING AND REMOVING ELEMENTS

As with the `Collection<T>` interface, the following methods are optional, which means they still must be implemented, but the implementation can simply throw an `UnsupportedOperationException` if for some reason you do not want to give them a "real" implementation. An `UnsupportedOperationException` is a `RunTimeException` and so is not required to be caught or declared in a `throws` clause.

```
public boolean add(T element) (Optional)
```

If `element` is not already in the calling object, `element` is added to the calling object and `true` is returned. If `element` is in the calling object, the calling object is unchanged and `false` is returned.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the class of `element` prevents it from being added to the set.
Throws a `NullPointerException` if `element` is `null` and the set does not support `null` elements.
Throws an `IllegalArgumentException` if some other aspect of `element` prevents it from being added to this set.

(continued)

Set<T>: Mètodes de la interfície

Display 16.3 Methods in the Set<T> Interface

```
public boolean addAll(Collection<? extends T> collectionToAdd) (Optional)
```

Ensures that the calling object contains all the elements in `collectionToAdd`. Returns `true` if the calling object changed as a result of the call; returns `false` otherwise. Thus, if `collectionToAdd` is a `Set<T>`, then the calling object is changed to the union of itself with `collectionToAdd`.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the class of some element of `collectionToAdd` prevents it from being added to the calling object.
Throws a `NullPointerException` if `collectionToAdd` contains one or more `null` elements and the calling object does not support `null` elements, or if `collectionToAdd` is `null`.
Throws an `IllegalArgumentException` if some aspect of some element of `collectionToAdd` prevents it from being added to the calling object.

(continued)

Set<T>: Mètodes de la interfície

Display 16.3 Methods in the Set<T> Interface

```
public boolean remove(Object element) (Optional)
```

Removes the `element` from the calling object, if it is present. Returns `true` if the calling object contained the `element`; returns `false` otherwise.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the type of `element` is incompatible with the calling object (optional).
Throws a `NullPointerException` if `element` is `null` and the calling object does not support `null` elements (optional).

(continued)

Set<T>: Mètodes de la interfície

Display 16.3 Methods in the Set<T> Interface

```
public boolean removeAll(Collection<?> collectionToRemove) (Optional)
```

Removes all the calling object's elements that are also contained in `collectionToRemove`. Returns `true` if the calling object was changed; otherwise returns `false`.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the types of one or more elements in `collectionToRemove` are incompatible with the calling object (optional).
Throws a `NullPointerException` if the calling object contains a `null` element and `collectionToRemove` does not support `null` elements (optional).
Throws a `NullPointerException` if `collectionToRemove` is `null`.

(continued)

Set<T>: Mètodes de la interfície

Display 16.3 Methods in the Set<T> Interface

`public void clear()` *(Optional)*

Removes all the elements from the calling object.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.

`public boolean retainAll(Collection<?> saveElements)` *(Optional)*

Retains only the elements in the calling object that are also contained in the collection `saveElements`. In other words, removes from the calling object all of its elements that are not contained in the collection `saveElements`. Returns `true` if the calling object was changed; otherwise returns `false`. If the argument is itself a `Set<T>`, this changes the calling object to the intersection of itself with the argument.
Throws an `UnsupportedOperationException` if this method is not supported by the class that implements this interface.
Throws a `ClassCastException` if the types of one or more elements in the calling object are incompatible with `saveElements` (optional).
Throws a `NullPointerException` if `saveElements` contains a `null` element and the calling object does not support `null` elements (optional).
Throws a `NullPointerException` if `saveElements` is `null`.

Implementacions de Set<T>

- Diverses implementacions de la interfície **`Set<T>:`**
  - **`TreeSet<T>:`** Implementació com a arbre binari balancejat.
  - **`HashSet<T>:`** *[Més usat]* Implementació amb taules de Hash
- Exemple:

```
Set s1 = new HashSet<Integer>();
Set s2 = new TreeSet();
```

Exemples de us del Set

```
Set set = new HashSet(); // instantiate a concrete set
// ...
set.add(obj); // insert an elements
// ...
int n = set.size(); // get size
// ...
if (set.contains(obj)) {...} // check membership

// iterate through the set
Iterator iter = set.iterator();
while (iter.hasNext()) {
  Object e = iter.next();
  // downcast e
  // ...
}
```

Col·leccions i tipus fonamentals

- Les col·leccions només poden contenir objectes.
  - No es pot posar un tipus de dades fonamental en una col·lecció
- Java ha definit "embolcalls" (***wrappers***) per a les classes fonamentals que tenen només valors de un tipus fonamental dins de l'objecte
- Aquestes classes es defineixen en java.lang
- Cada tipus de dades fonamental està representada per una classe contenidora (wrapper). Les classes wrapper són:
  - Boolean
  - Byte
  - Character
  - Double
  - Float
  - Short
  - Integer
  - Long

Col·leccions i tipus fonamentals

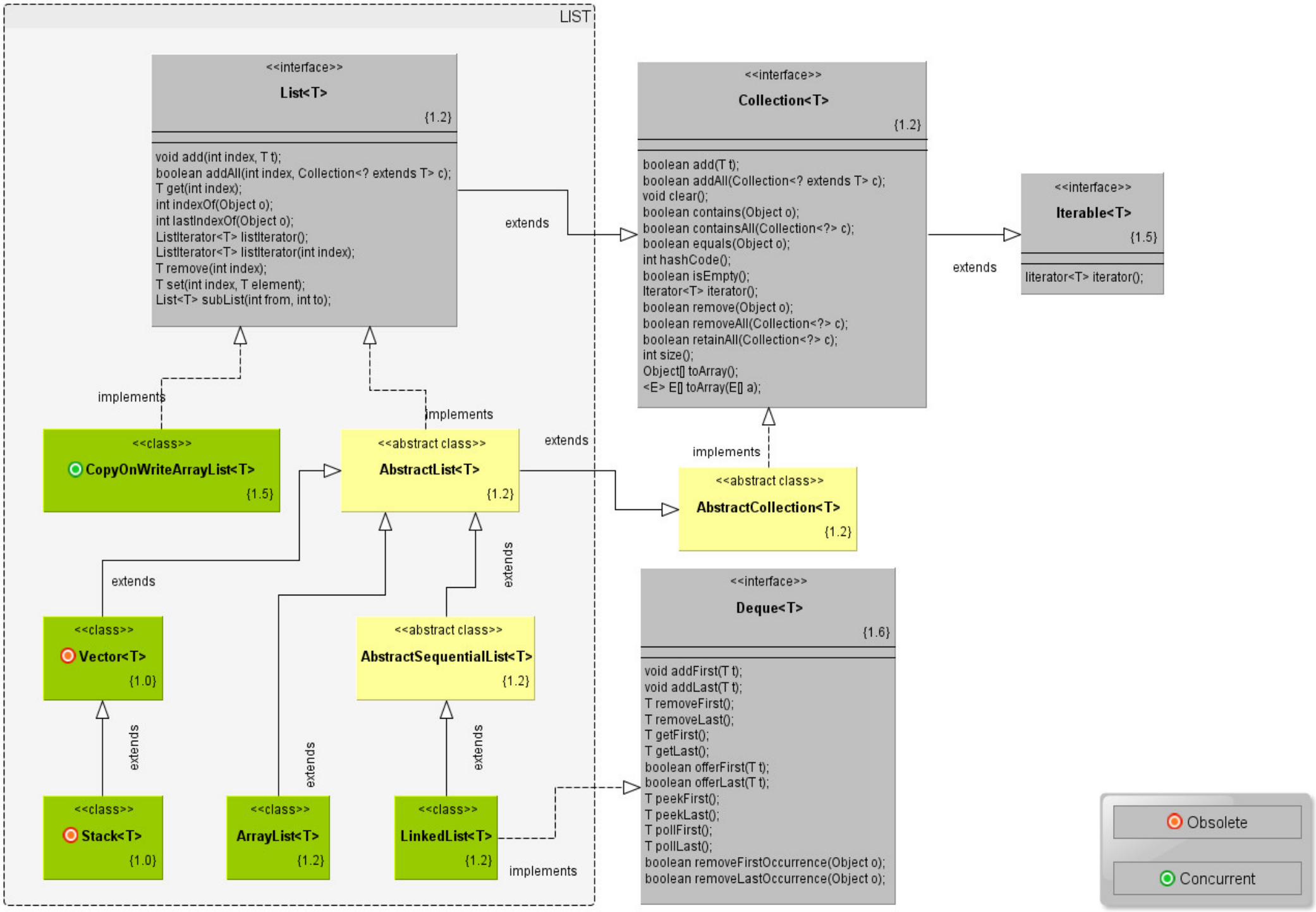
- Les classes de contenidor s'utilitzen generalment perquè els valors de dades fonamentals es poden colocar dins d'una col·lecció.
- Les classes contenidores tenen variables útils de classe. Integer.MAX_VALUE, Integer.MIN_VALUE Double.MAX_VALUE, Double.MIN_VALUE, Double.NaN, Double.NEGATIVE_INFINITY, Double.POSITIVE_INFINITY
- També tenen mètodes útils de classe Double.parseDouble (String) - converteix una cadena a un doble Integer.parseInt (String) - converteix una cadena a un enter

Llistes

Una implementació de les col·leccions

Llistes: List<T>

- Les classes que implementen la interfície **`List<T>`** tenen els seus elements ordenats
- Els elements s'indexen començant per zero
- Una classe que implementa la interfície **`List<T>`** permet que els elements apareguin més d'una vegada
- La interfície **`List<T>`** té més funcions que els mètodes de la interfície **`Collection<T>`**
- Alguns dels mètodes heretats de la interfície **`Collection<T>`** tenen semàntiques diferents en la interfície **`List<T>`**


LIST
<<interface>>
List<T>
{1.2}
void add(int index, T t);
boolean addAll(int index, Collection<? extends T> c);
T get(int index);
int indexOf(Object o);
int lastIndexOf(Object o);
ListIterator<T> listIterator();
ListIterator<T> listIterator(int index);
T remove(int index);
T set(int index, T element);
List<T> subList(int from, int to);
extends
<<interface>>
Collection<T>
{1.2}
boolean add(T t);
boolean addAll(Collection<? extends T> c);
void clear();
boolean contains(Object o);
boolean containsAll(Collection<?> c);
boolean equals(Object o);
int hashCode();
boolean isEmpty();
Iterator<T> iterator();
boolean remove(Object o);
boolean removeAll(Collection<?> c);
boolean retainAll(Collection<?> c);
int size();
Object[] toArray();
<E> E[] toArray(E[] a);
extends
<<interface>>
Iterable<T>
{1.5}
Iterator<T> iterator();
implements
implements
<<class>>
CopyOnWriteArrayList<T>
{1.5}
<<abstract class>>
AbstractList<T>
{1.2}
extends
implements
<<abstract class>>
AbstractCollection<T>
{1.2}
extends
extends
<<class>>
Vector<T>
{1.0}
<<abstract class>>
AbstractSequentialList<T>
{1.2}
<<interface>>
Deque<T>
{1.6}
void addFirst(T t);
void addLast(T t);
T removeFirst();
T removeLast();
T getFirst();
T getLast();
boolean offerFirst(T t);
boolean offerLast(T t);
T peekFirst();
T peekLast();
T pollFirst();
T pollLast();
boolean removeFirstOccurrence(Object o);
boolean removeLastOccurrence(Object o);
extends
extends
extends
<<class>>
Stack<T>
{1.0}
<<class>>
ArrayList<T>
{1.2}
<<class>>
LinkedList<T>
{1.2}
implements
Obsolete
Concurrent

List<T>: Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

The `List<T>` interface is in the `java.util` package.
The `List<T>` interface extends the `Collection<T>` interface.
All the exception classes mentioned are the kind that are not required to be caught in a `catch` block or declared in a `throws` clause.
All the exception classes mentioned are in the package `java.lang` and so do not require any import statement.

CONSTRUCTORS

Although not officially required by the interface, any class that implements the `List<T>` interface should have at least two constructors: a no-argument constructor that creates an empty `List<T>` object, and a constructor with one parameter of type `Collection<? extends T>` that creates a `List<T>` object with the same elements as the constructor argument. If the argument imposes an ordering on its elements, then the `List<T>` created should preserve this ordering.

```
boolean isEmpty()
```

Returns `true` if the calling object is empty; otherwise returns `false`.

(continued)

List<T>: Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

`public boolean contains(Object target)`

Returns `true` if the calling object contains at least one instance of `target`. Uses `target.equals` to determine if `target` is in the calling object.
Throws a `ClassCastException` if the type of `target` is incompatible with the calling object (optional).
Throws a `NullPointerException` if `target` is `null` and the calling object does not support `null` elements (optional).

`public boolean containsAll(Collection<?> collectionOfTargets)`

Returns `true` if the calling object contains all of the elements in `collectionOfTargets`. For an `element` in `collectionOfTargets`, this method uses `element.equals` to determine if `element` is in the calling object. The elements need not be in the same order or have the same multiplicity in `collectionOfTargets` and in the calling object.
Throws a `ClassCastException` if the types of one or more elements in `collectionOfTargets` are incompatible with the calling object (optional).
Throws a `NullPointerException` if `collectionOfTargets` contains one or more `null` elements and the calling object does not support `null` elements (optional).
Throws a `NullPointerException` if `collectionOfTargets` is `null`.

(continued)

List<T>: Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

```
public boolean equals(Object other)
```

If the argument is a `List<T>`, returns `true` if the calling object and the argument contain exactly the same elements in exactly the same order; otherwise returns `false`. If the argument is not a `List<T>`, `false` is returned.

```
public int size()
```

Returns the number of elements in the calling object. If the calling object contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.

```
Iterator<T> iterator()
```

Returns an iterator for the calling object. (Iterators are discussed in Section 16.2.)

(continued)

List<T>: Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

```
public Object[] toArray()
```

Returns an array containing all of the elements in the calling object. The elements in the returned array are in the same order as in the calling object. A new array must be returned so that the calling object has no references to the returned array.

List<T>: Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

`public boolean addAll(Collection<? extends T> collectionToAdd)` *(Optional)*

Adds all of the elements in `collectionToAdd` to the end of the calling object's list. The elements are added in the order they are produced by an iterator for `collectionToAdd`.
Throws an `UnsupportedOperationException` if the `addAll` method is not supported by the calling object.
Throws a `ClassCastException` if the class of an element in `collectionToAdd` prevents it from being added to the calling object.
Throws a `NullPointerException` if `collectionToAdd` contains one or more `null` elements and the calling object does not support `null` elements, or if `collectionToAdd` is `null`.
Throws an `IllegalArgumentException` if some aspect of an element in `collectionToAdd` prevents it from being added to the calling object.

`public boolean remove(Object element)` *(Optional)*

Removes the first occurrence of `element` from the calling object's list, if it is present. Returns `true` if the calling object contained the `element`; returns `false` otherwise.
Throws a `ClassCastException` if the type of `element` is incompatible with the calling object (optional).
Throws a `NullPointerException` if `element` is `null` and the calling object does not support `null` elements (optional).
Throws an `UnsupportedOperationException` if the `remove` method is not supported by the calling object.

(continued)

List<T>: Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

`public boolean removeAll(Collection<?> collectionToRemove)` *(Optional)*

Removes all the calling object's elements that are also in `collectionToRemove`. Returns `true` if the calling object was changed; otherwise returns `false`.
Throws an `UnsupportedOperationException` if the `removeAll` method is not supported by the calling object.
Throws a `ClassCastException` if the types of one or more elements in the calling object are incompatible with `collectionToRemove` (optional).
Throws a `NullPointerException` if the calling object contains one or more `null` elements and `collectionToRemove` does not support `null` elements (optional).
Throws a `NullPointerException` if `collectionToRemove` is `null`.

`public void clear()` *(Optional)*

Removes all the elements from the calling object.
Throws an `UnsupportedOperationException` if the `clear` method is not supported by the calling object.

(continued)

List<T>: Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

`public boolean retainAll(Collection<?> saveElements)` *(Optional)*

Retains only the elements in the calling object that are also in the collection `saveElements`. In other words, removes from the calling object all of its elements that are not contained in the collection `saveElements`. Returns `true` if the calling object was changed; otherwise returns `false`.
Throws an `UnsupportedOperationException` if the `retainAll` method is not supported by the calling object.
Throws a `ClassCastException` if the types of one or more elements in the calling object are incompatible with `saveElements` (optional).
Throws a `NullPointerException` if the calling object contains one or more `null` elements and `saveElements` does not support `null` elements (optional).
Throws a `NullPointerException` if the `saveElements` is `null`.

NEW METHOD HEADINGS

The following methods are in the `List<T>` interface but were not in the `Collection<T>` interface. Those that are optional are noted.

(continued)

List<T>: NOUS Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

`public void add(int index, T newElement)` *(Optional)*

Inserts `newElement` in the calling object's list at location `index`. The old elements at location `index` and higher are moved to higher indices.
Throws an `IndexOutOfBoundsException` if the `index` is not in the range:

```
0 <= index <= size()
```

Throws an `UnsupportedOperationException` if this `add` method is not supported by the calling object.
Throws a `ClassCastException` if the class of `newElement` prevents it from being added to the calling object.
Throws a `NullPointerException` if `newElement` is `null` and the calling object does not support `null` elements.
Throws an `IllegalArgumentException` if some aspect of `newElement` prevents it from being added to the calling object.

(continued)

List<T>: NOUS Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

```
public boolean addAll(int index,
                      Collection<? extends T> collectionToAdd) (Optional)
```

Inserts all of the elements in `collectionToAdd` to the calling object's list starting at location `index`. The old elements at location `index` and higher are moved to higher indices. The elements are added in the order they are produced by an iterator for `collectionToAdd`.
Throws an `IndexOutOfBoundsException` if the `index` is not in the range:

```
0 <= index <= size()
```

Throws an `UnsupportedOperationException` if the `addAll` method is not supported by the calling object.
Throws a `ClassCastException` if the class of one of the elements of `collectionToAdd` prevents it from being added to the calling object.
Throws a `NullPointerException` if `collectionToAdd` contains one or more `null` elements and the calling object does not support `null` elements, or if `collectionToAdd` is `null`.
Throws an `IllegalArgumentException` if some aspect of one of the elements of `collectionToAdd` prevents it from being added to the calling object.

(continued)

List<T>: NOUS Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

```
public T get(int index)
```

Returns the object at position `index`.
Throws an `IndexOutOfBoundsException` if the `index` is not in the range:

```
0 <= index < size()
```

```
public T set(int index, T newElement) (Optional)
```

Sets the element at the specified `index` to `newElement`. The element previously at that position is returned.
Throws an `IndexOutOfBoundsException` if the `index` is not in the range:

```
0 <= index < size()
```

Throws an `UnsupportedOperationException` if the `set` method is not supported by the calling object.
Throws a `ClassCastException` if the class of `newElement` prevents it from being added to the calling object.
Throws a `NullPointerException` if `newElement` is `null` and the calling object does not support `null` elements.
Throws an `IllegalArgumentException` if some aspect of `newElement` prevents it from being added to the calling object.

(continued)

List<T>: NOUS Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

`public T remove(int index)` *(Optional)*

Removes the element at position `index` in the calling object. Shifts any subsequent elements to the left (subtracts one from their indices). Returns the element that was removed from the calling object.
Throws an `UnsupportedOperationException` if the `remove` method is not supported by the calling object.
Throws an `IndexOutOfBoundsException` if `index` does not satisfy:

```
0 <= index < size()
```

(continued)

List<T>: NOUS Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

`public int indexOf(Object target)`

Returns the index of the first element that is equal to `target`. Uses the method `equals` of the object `target` to test for equality. Returns `–1` if `target` is not found.
Throws a `ClassCastException` if the type of `target` is incompatible with the calling object (optional).
Throws a `NullPointerException` if `target` is `null` and the calling object does not support `null` elements (optional).

`public int lastIndexOf(Object target)`

Returns the index of the last element that is equal to `target`. Uses the method `equals` of the object `target` to test for equality. Returns `–1` if `target` is not found.
Throws a `ClassCastException` if the type of `target` is incompatible with the calling object (optional).
Throws a `NullPointerException` if `target` is `null` and the calling object does not support `null` elements (optional).

(continued)

List<T>: NOUS Mètodes de la interfície

Display 16.4 Methods in the `List<T>` Interface

```
public List<T> subList(int fromIndex, int toIndex)
```

Returns a *view* of the elements at locations `fromIndex` to `toIndex` of the calling object; the object at `fromIndex` is included; the object, if any, at `toIndex` is not included. The *view* uses references into the calling object; so, changing the view can change the calling object. The returned object will be of type `List<T>` but need not be of the same type as the calling object. Returns an empty `List<T>` if `fromIndex` equals `toIndex`.
Throws an `IndexOutOfBoundsException` if `fromIndex` and `toIndex` do not satisfy:

```
0 <= fromIndex <= toIndex <= size()
```

(continued)

Implementacions de List<T>

- Diverses implementacions de la interfície **`List<T>:`**
  - **`ArrayList<T>:`** Implementació de la llista amb una taula redimensionable. Accés més rapid.
  - **`LinkedList<T>:`** Implementació de la llista amb punters. Insercions i eliminacions més ràpides
- Exemple:

```
List<String> noms = new LinkedList<String>;
```

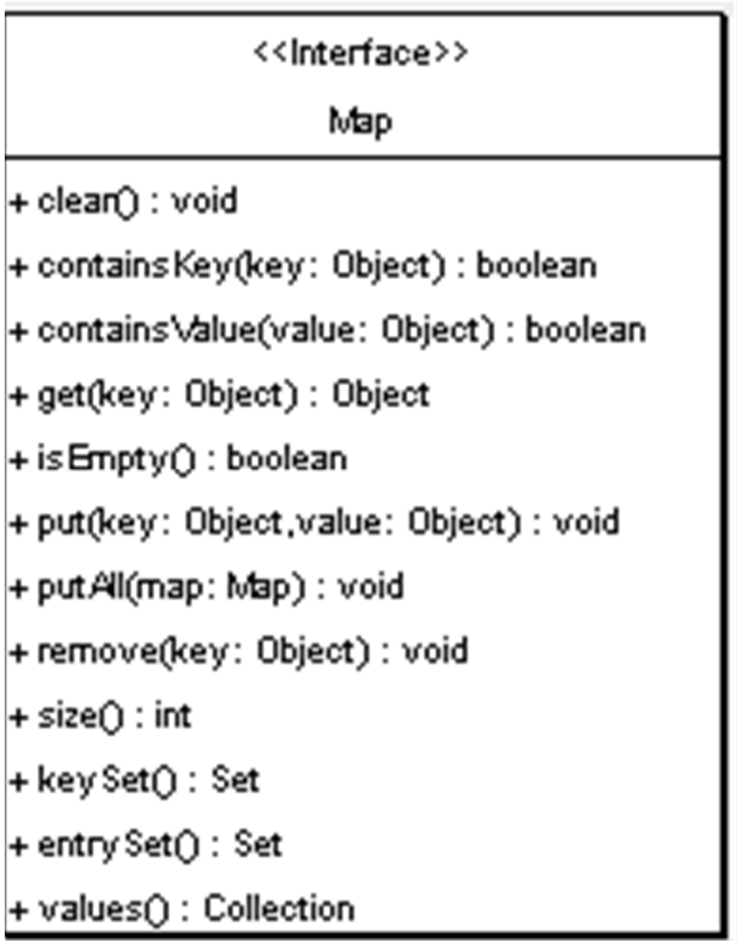
Maps

Una agrupació d'objectes que no és una col·lecció

Maps

- Implementació de memòria associativa
- Parelles <clau,valor>
- S'obté el valor associat a una clau no pel índex sinó per un càlcul (de hash) sobre la clau.
- Per exemple. Volem guardar freqüències de paraules
  - <String, Integer>
  - Accedim a nombre de vegades que apareix la paraula per la mateixa paraula (no per un índex com en una taula)

Map<K,V>: Mètodes de la interfície



Mètodes de la interfície

Clear()	Elimina totes les associacions
containsKey(k)	Si conté una assoc. per a k
containsValue(v)	Si conté una assoc. per a v
get(k)	Valor associat amb k
isEmpty()	Si esta buit
keySet()	Conjunt de claus
put(k,v)	Associar v amb k
remove(k)	Eliminar associacions per a k
size()	Nombre de parelles
values()	Col·lecció de valors

Implementacions de Map<K,V>

- Diverses implementacions de la interfície **`Map<T>:`**
  - **`HashMap:`** Implementació del Map amb taules de hast. *[Més usat]*
  - **`TreeMap:`** Implementació del Map amb arbres balancejats. Manté un ordre.
- Exemple:

```
Map<String,Integer> noms = new HashMap<String,Integer>;
Map map = new HashMap();
```

Exemple 1 de us del Map<K,V>

```
Map map = new HashMap(); // instantiate a concrete map
// ...
map.put(key, val); // insert a key-value pair
// ...
// get the value associated with key
Object val = map.get(key);
map.remove(key); // remove a key-value pair
// ...
if (map.containsValue(val)) { ... }
if (map.containsKey(kay)) { ... }
Set keys = map.keySet(); // get the set of keys
// iterate through the set of keys
Iterator iter = keys.iterator();
while (iter.hasNext()) {
  Key key = (Key) iter.next();
  // ...
}
```

Exemple 2 de us del Map<K,V>

```
HashMap<String, Integer> frequency(String[] names) {

    HashMap<String, Integer> frequency =
                new HashMap<String, Integer>();

    for(String name : names) {
        Integer currentCount = frequency.get(name);
        if(currentCount == null) {
            currentCount = 0; }
        frequency.put(name, ++currentCount);
    }
    return frequency;
}
```

Us del Map per a classes

- Per poder fer servir el Map, la classe que es fa servir com a clau ha de tenir implementats de els mètodes:

  `equals`
  `hashCode`

- En les classes habituals estan implementades, en les pròpies s'han de implementar

Us del Map per a classes

```
public class Person {
    public String name;
    boolean equals(Object o) {
        return (o instanceof Person && ((Person)o).name.equals(name));
    }
    public int hashCode() {
        return name.hashCode();
    }
}
```

Iterations

Recorrent les col·leccions

Iteradors. Recorrent les col·leccions

- Les col·leccions i els Maps són extensions de la classe iterables.
- A la classe hi ha mètodes definits per recórrer els elements que componen l'objecte de la classe

Interfície Iteration

- Mètodes implementats

```
boolean hasNext( );
Object next( );
void remove( );
```

- Mètodes implementats addicionals pels ListIteration

```
boolean hasPrevious();
Object previous();
int nextIndex( );
int previousIndex( );
```

Us dels iteradors

- Exemple de mètode que imprimeix tots els elements d'una col·lecció:

```
static void printAll (Collection coll) {
 Iterator iter = coll.iterator( );
 while (iter.hasNext( )) {
    System.out.println(iter.next( ) );
 }
}
```

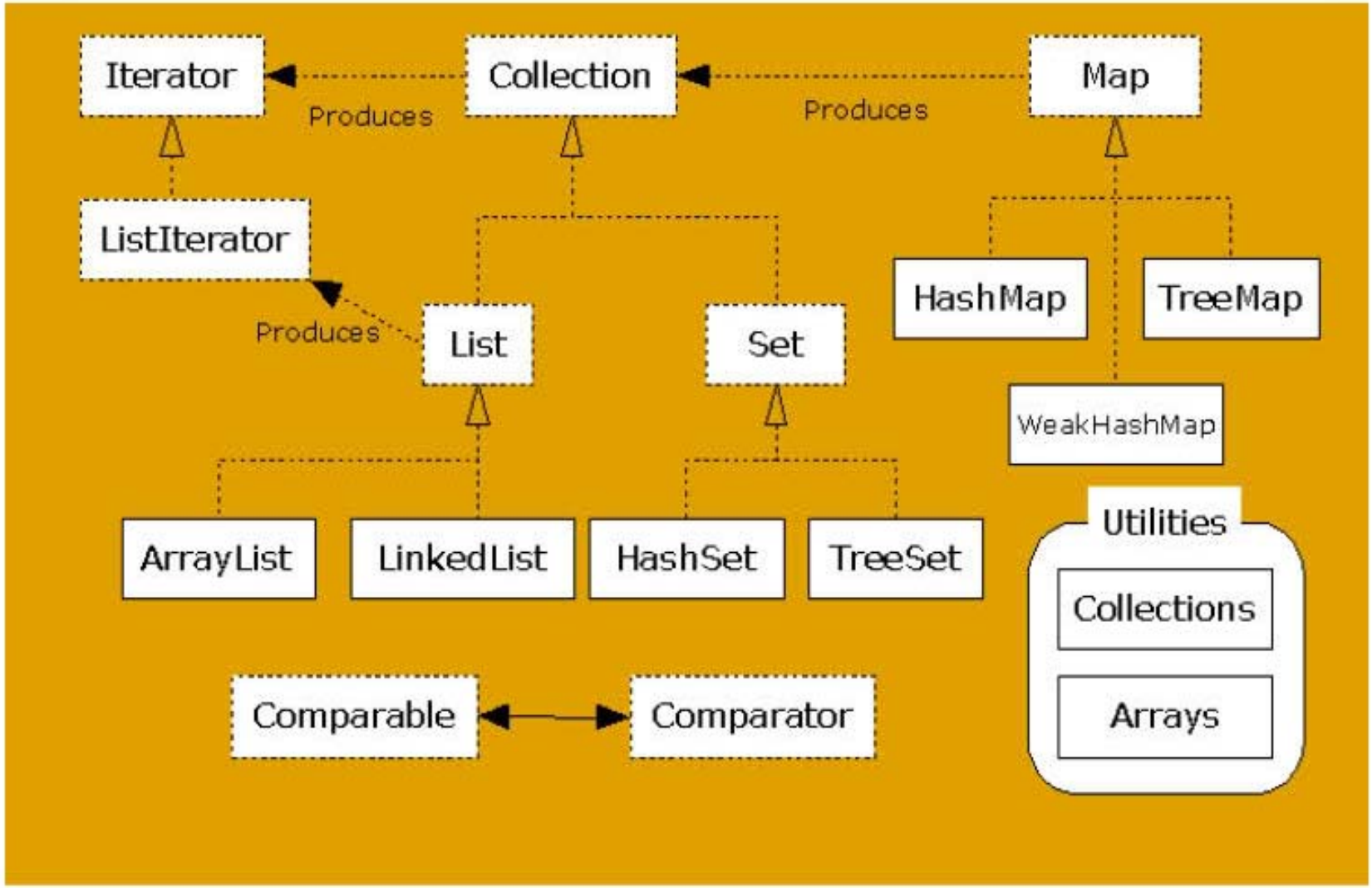
- Observeu que el codi és polimòrfic, és a dir, que funcionarà per qualsevol col·lecció.
- Hi ha constructores de iteradors específics per llistes:

```
ListIterator listIterator( );
ListIterator listIterator(int index);
```

la darrera comença en la posició *index*

Resum

Resum visual



Altres agrupacions

- Cues
- Piles
- ...

public interface Queue<E> extends Collection<E>

```
public interface Queue<E> extends
  Collection<E> {
     E element();              //throws
     E peek();                 //null
     boolean offer(E e);       //add - bool
     E remove();               //throws
     E poll();                 //null
}
```

Collections – Other Functions

- La classe `java.util.Collections` té molts mètodes útils per treballar amb col·leccions
- Pràcticament tots els algorismes requereixen implementar pels teus objectes una interfície addicional, anomenat `Comparable` o `Comparator`.

Algorismes disponibles per llistes

- Són mètodes estàtics que implementen algorismes usuals en llistes:
  - sort: Ordena fent servir mergesort
  - shuffle: Barreja aleatòriament els elements de la llista
  - reverse: Inverteix l'ordre
  - rotate: Rota la llista el nombre de vegades que vulguem
  - fill: Reescriu tots els elements de la llista amb l'element que volem.
  - copy: Copia una llista sobre una altra
  - binarySearch: Cerca fent servir cerca binaria
  - indexOfSubList: Retorna el índex de la primera subllista trobada
  - lastIndexOfSubList: Retorna el índex de la darrera subllista trobada
  - max: Retorna el màxim de la llista
  - min: Retorna el mínim de la llista

Comparator

- Per ordenar, trobar un element, trobar el màxim, etc, es necessita comparar objectes.
- Definirem doncs un mètode comparador de objectes, de manera que donats 2 elements retorni un enter:
  - Negatiu: quan el primer sigui menor que el segon
  - Zero: quan siguin iguals
  - Positiu

Comparator

- S'implementa la classe Comparator amb l'únic mètode que conté, amb la funció de comparació que volem fer servir
- Utilitzem la funció de comparació passant una instancia de la classe definida al mètode de ordenació o el que sigui.

Implementació classe Comparator: definició del mètode compare

```
public class CountComparator implements Comparator {
  public int compare(Object o1, Object o2) {
    if (o1 != null &&
        o2 != null &&
        o1 instanceof Count &&
        o2 instanceof Count) {
      Count c1 = (Count) o1;
      Count c2 = (Count) o2;
      return (c2.i - c1.i);
    } else {
      return 0;
    }
  }
}
```

Us de comparador

```
public class WordFrequency4 {
  static public void main(String[] args) {
    ....
    List list = new ArrayList(words.values());
    Collections.sort(list, new CountComparator());
    Iterator iter = list.iterator();
    while (iter.hasNext()) {
      count = (Count) iter.next();
      word = count.word;
      System.out.println(word +
          (word.length() < 8 ? "\t\t" : "\t") +
          count.i);
    }
  }
}
```