Colecciones e iteradores

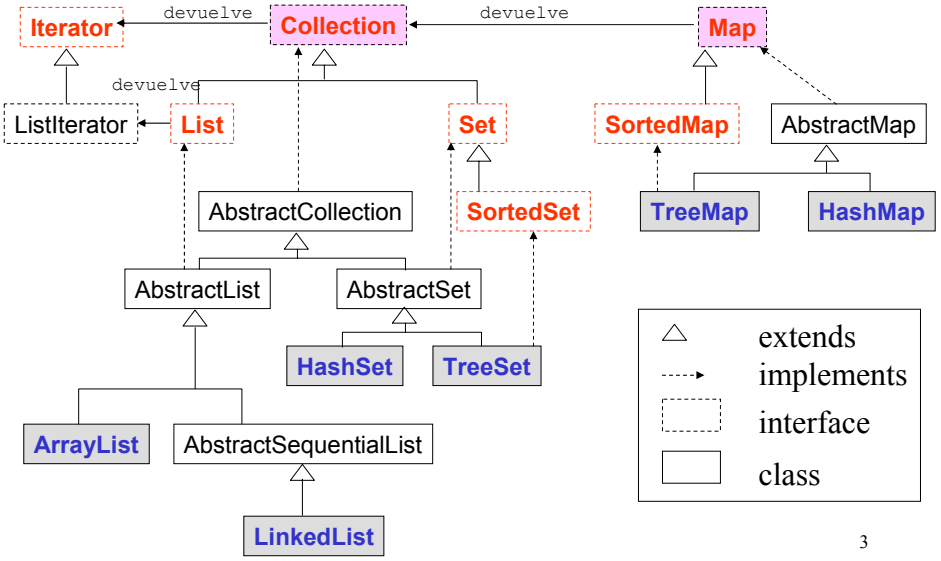
- interfaz `Collection`
  - clases `ArrayList`, `LinkedList`, `HashSet`, `TreeSet`
- interfaz `Map`
  - clases `TreeMap`, `HashMap`
- Iteratores: interfaz `Iterator`

Colecciones en Java

- Permite *almacenar* y organizar *objetos* de manera útil para un acceso eficiente.
- Se encuentran en el paquete **`java.util`**
- Núcleo de abstracciones de colecciones de utilidad (interfaces) e implementaciones ampliamente útiles.
- Las interfaces proporcionan métodos para todas las operaciones comunes y las implementaciones concretas especifican la decisión de las operaciones no permitidas.

  (`java.lang.UnsupportedOperationException`)
- Sobre los elementos se puede iterar (**`Iterator`**)

Jerarquía de colecciones



Interfaz Collection (1/2)

- **Collection<T>**
  - int **size**()
  - boolean **empty**()
  - boolean **contains**(Object elem)
  - Iterator<T> **iterator**()
  - Object[] **toArray**
  - boolean **add**(T elem),
  - boolean **remove**(Object elem)
  - void **clear**()

- **List<T>** – Una colección cuyos elementos permanecen en un orden particular a menos que se modifique la lista (no significa lista enlazada aunque es una posible implementación).
  - void **add**(int index, T element)
  - T **remove**(int index)
  - T **get**(int index)
  - T **set**(int index, T element)
  - int **indexOf**(Object o)
  - int **lastIndexOf**(Object o)
  - List<T> **subList**(int min, int max)

Interfaz Collection (2/2)

-Set<T> – Una colección (conjunto) donde no puede haber elementos repetidos, y cuyos elementos no se almacenan necesariamente siguiendo un orden particular.

- Mismos métodos que `Collection` con otro contrato.

-SortedSet<T> – Conjunto con elementos ordenados.

```
-T first()
-T last()
-SortedSet<T> subSet(T fromElement, T toElement)
-SortedSet<T> headSet(T toElement)
-SortedSet<T> tailSet(T fromElement)
```


Interfaz Map

- **Map<K,V>**
  - Un objeto que asocia claves con valores.
  - No puede tener claves duplicadas.
  - `Object put(K key, V value); V remove(Object key); V get(Object key);`
  - `containsKey, containsValue, isEmpty, size`
  - `Set<K> keySet()` //Colección de claves
  - `Collection<V> values()` //Colección de valores
  - `Set<Map.Entry<K,V>> entrySet()` //Colección de pares clave-valor
- **SortedMap<K,V>:** Un mapa cuyas claves están ordenadas.
  - `K firstKey()`
  - `K lastKey()`
  - `SortedMap<K,V> subMap(K minKey, K maxKey)`
  - `SortedMap<K,V> headMap(K maxKey)`
  - `SortedMap<K,V> tailMap(K minKey)`

Entrada = <clave, valor>

- **`Map.Entry`** clase interna de `java.util.Map`

```
java.util.Map.Entry<K,V>{
   //proporciona la clave de la entrada
   K getKey();
   //proporciona el valor de la entrada
   V getValor();
   //cambia el valor de la entrada
   //devuelve el valor antiguo
   V setValue(V nuevoValor);
}
```


Iteradores

- `public interface Iterable<T>{`
  `Iterator<T> iterator();`
  `}`

- `interface Collection<T> implements Iterable<T>`
  - Se puede iterar sobre cualquier colección
  - Se puede utilizar el bucle “for each” para cualquier colección

- `interface Iterator<T>{`

```
   boolean hasNext();
   T next();
   void remove();
}
```

Iteración

```
interface Iterator<T>{
      /** Devuelve true si la iteración tiene mas elementos */
      boolean hasNext();

      /** Devuelve el siguiente elemento de la iteración
      /* @throws NoSuchElementException si se ha llegado al
      /* final de la colección
      */
      T next();

      /** Elimina el último elemento devuelto por la iteración
      /* @throws IllegalStateException si se llama
      /* a remove sin haber llamado primero a next
      */
      void remove();
      /* si no lo implementa lanzamos UnsupportedOperationException */
}
```


Ejemplo de uso de Iteradores

- Cálculo del gasto total de un departamento

```
public double gastoDpto(){
   double gasto=0;
   Iterator<Empleado> it = plantilla.iterator();
   while (it.hasNext()){
       Empleado e = it.next();
       gasto += e.getSueldo();
   }
   return gasto;
}
```

Siendo plantilla una colección que implemente la interfaz `Collection`
Por ejemplo **`LinkedList<Empleado> plantilla;`**

Iterar sobre listas

```
interface ListIterator<T> extends Iterator<T>{
   //añade elem delante de la posición actual
   void add (T elem);
   //reemplaza el último elemento visitado por next o
   //previous por elem
   void set (T elem);
   //true si hay otro elementos que visitar en un recorrido
   //hacia atrás
   boolean hasPrevious();
   //objeto anterior
   T previous();
   //devuelve el índice del elemento que proporcionaría
   //la siguiente llamada a next
   int nextIndex();
   //devuelve el índice del elemento que proporcionaría
   //la siguiente llamada a previous
   int previousIndex();
}
```


Implementaciones de `Collection`

`LinkedList<T>` – Una implementación de una lista doblemente enlazada. La **modificación** es **poco costosa** para cualquier tamaño, pero el **acceso aleatorio** es **lento**. Útil para implementar colas y pilas.

`-getFirst, getLast, removeFirst, removeLast, addFirst, addLast`

`ArrayList<T>` – Una lista implementada utilizando un array de dimensión modificable. Es **costoso añadir o borrar** un elemento cerca del principio de la lista si ésta es grande, pero es relativamente poco costoso de crear y **rápido** para **acceso aleatorio**.

`HashSet<T>` – Un `Set` implementado mediante una tabla *hash*. Es una buena implementación de propósito general por lo que la búsqueda, la adición y eliminación son **insensibles al tamaño** de los contenidos.

`TreeSet<T>` – Un `SortedSet` implementado utilizando un árbol binario equilibrado. Es **más lento para buscar o modificar que un `HashSet`**, pero mantiene los elementos ordenados. Asume que los elementos son *comparables* si no se le ha pasado un *comparator* en el constructor.

Convenciones sobre excepciones

- **`UnsupportedOperationException`**
  - Métodos opcionales en la implementación de una interfaz
- **`ClassCastException`**
  - El tipo del elemento que se desea insertar no es del tipo apropiado
- **`IllegalArgumentException`**
  - El valor del elemento no es apropiado para la colección
- **`NoSuchElementException`**
  - La colección de la que se quiere devolver un elemento está vacía
- **`NullPointerException`**
  - Se pasa como argumento una referencia con valor `null` cuando la colección no admite este valor.

Declaración y recorrido de colecciones

```
import java.util.*;

public class TestListas {
  public static void main( String args[] ) {
    List<Integer> lista = new ArrayList<Integer>();

     for( int i=0; i < Integer.parseInt(args[0]); i++ )
            lista.add(i);

    //recorrido con iterador
     Iterator<Integer> it = lista.iterator();
      while( it.hasNext() )
            System.out.println(it.next());

    //recorrido bucle "for each"
      for (Integer ent : lista)
             System.out.println(ent);
  }
}
```

interfaz

Clase concreta

autoboxing

Implementaciones de `Map`

- **`HashMap<K,V>`**
  - Una implementación de `Map` con una *tabla hash*.
  - El método `hashCode` de cada clave se utiliza para seleccionar un lugar en la tabla
  - Una colección de utilidad muy general con tiempos relativamente cortos de búsqueda e inserción.

- **`TreeMap<K,V>`**
  – Una implementación de `SortedMap` utilizando un árbol binario equilibrado que mantiene sus elementos ordenados por clave.
  – Útil para conjuntos de datos ordenados que requieren una búsqueda por clave moderadamente rápida.
  -Asume que los elementos son *comparables* si no se le ha pasado un *comparator* en el constructor.

Ejemplo 1/2

```
public class TestMapa{
 public static void main(String [] args){
   Map<String,Empleado> plantilla =
                                  new HashMap<String,Empleado>();
   plantilla.put("34806408V", new Empleado("Pedro Mtnez"));
   plantilla.put("34186581A", new Empleado("Pablo Fdez"));
   plantilla.remove("34806408V");
   System.out.println(plantilla.get("34186581A"));
       //recorrer todas las entradas del mapa
   for (Map.Entry<String,Empleado> entrada : plantilla.entrySet())
   {
       String clave = entrada.getKey();
       Empleado e = entrada.getValue();
       System.out.println("clave = "+clave+", valor = "+e);
   }
 }
}
```

Las utilidades de `Collections (v1.4)`

- `public static Object` **`min`**`(Collection col)`
- `public static Object` **`max`**`(Collection col)`
- `public static Object` **`min`**`(Collection col, Comparator comp)`
- `public static Object` **`max`**`(Collection col, Comparator comp)`
- `public static void` **`reverse`**`(List lista)`
- `public static void` **`copy`**`(List dst, List fnt)`
- `public static void` **`sort`**`(List lista)`
- `public static void` **`sort`**`(List lista, Comparator comp)`
- `public static int` **`binarySearch`**`(List lista, Object clave)`
- `public static int` **`binarySearch`**`(List lista, Object clave, Comparator comp)`