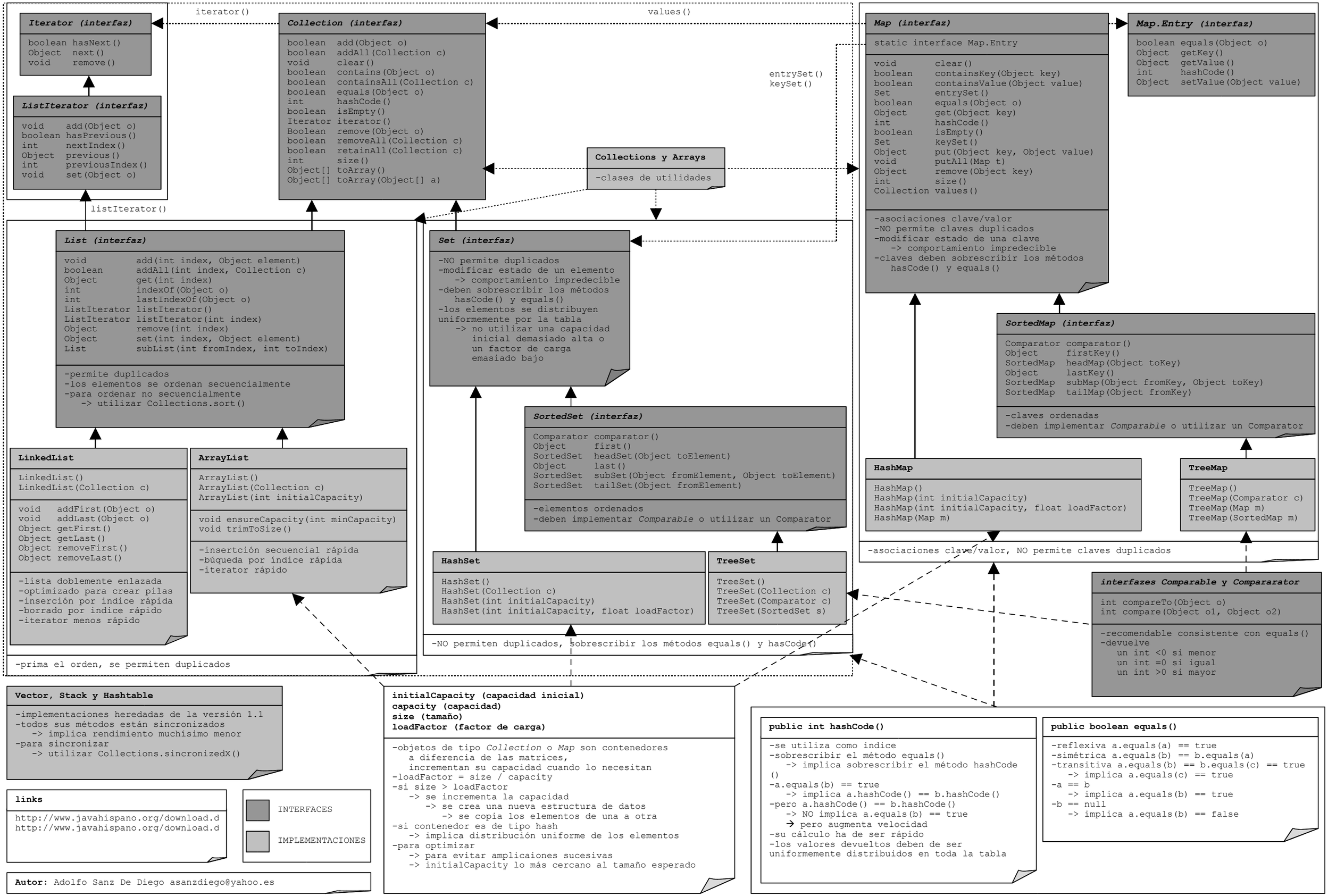
iterator()
values()
entrySet()
keySet()
listIterator()
Iterator (interfaz)
boolean hasNext()
Object next()
void remove()
ListIterator (interfaz)
void add(Object o)
boolean hasPrevious()
int nextIndex()
Object previous()
int previousIndex()
void set(Object o)
Collection (interfaz)
boolean add(Object o)
boolean addAll(Collection c)
void clear()
boolean contains(Object o)
boolean containsAll(Collection c)
boolean equals(Object o)
int hashCode()
boolean isEmpty()
Iterator iterator()
Boolean remove(Object o)
boolean removeAll(Collection c)
boolean retainAll(Collection c)
int size()
Object[] toArray()
Object[] toArray(Object[] a)
Collections y Arrays
-clases de utilidades
Map (interfaz)
static interface Map.Entry
void clear()
boolean containsKey(Object key)
boolean containsValue(Object value)
Set entrySet()
boolean equals(Object o)
Object get(Object key)
int hashCode()
boolean isEmpty()
Set keySet()
Object put(Object key, Object value)
void putAll(Map t)
Object remove(Object key)
int size()
Collection values()
-asociaciones clave/valor
-NO permite claves duplicados
-modificar estado de una clave
-> comportamiento impredecible
-claves deben sobrescribir los métodos
hasCode() y equals()
Map.Entry (interfaz)
boolean equals(Object o)
Object getKey()
Object getValue()
int hashCode()
Object setValue(Object value)
List (interfaz)
void add(int index, Object element)
boolean addAll(int index, Collection c)
Object get(int index)
int indexOf(Object o)
int lastIndexOf(Object o)
ListIterator listIterator()
ListIterator listIterator(int index)
Object remove(int index)
Object set(int index, Object element)
List subList(int fromIndex, int toIndex)
-permite duplicados
-los elementos se ordenan secuencialmente
-para ordenar no secuencialmente
-> utilizar Collections.sort()
Set (interfaz)
-NO permite duplicados
-modificar estado de un elemento
-> comportamiento impredecible
-deben sobrescribir los métodos
hasCode() y equals()
-los elementos se distribuyen
uniformemente por la tabla
-> no utilizar una capacidad
inicial demasiado alta o
un factor de carga
emasiado bajo
SortedMap (interfaz)
Comparator comparator()
Object firstKey()
SortedMap headMap(Object toKey)
Object lastKey()
SortedMap subMap(Object fromKey, Object toKey)
SortedMap tailMap(Object fromKey)
-claves ordenadas
-deben implementar Comparable o utilizar un Comparator
SortedSet (interfaz)
Comparator comparator()
Object first()
SortedSet headSet(Object toElement)
Object last()
SortedSet subSet(Object fromElement, Object toElement)
SortedSet tailSet(Object fromElement)
-elementos ordenados
-deben implementar Comparable o utilizar un Comparator
LinkedList
LinkedList()
LinkedList(Collection c)
void addFirst(Object o)
void addLast(Object o)
Object getFirst()
Object getLast()
Object removeFirst()
Object removeLast()
-lista doblemente enlazada
-optimizado para crear pilas
-inserción por indice rápida
-borrado por indice rápido
-iterator menos rápido
ArrayList
ArrayList()
ArrayList(Collection c)
ArrayList(int initialCapacity)
void ensureCapacity(int minCapacity)
void trimToSize()
-insertción secuencial rápida
-búqueda por índice rápida
-iterator rápido
-prima el orden, se permiten duplicados
HashSet
HashSet()
HashSet(Collection c)
HashSet(int initialCapacity)
HashSet(int initialCapacity, float loadFactor)
TreeSet
TreeSet()
TreeSet(Collection c)
TreeSet(Comparator c)
TreeSet(SortedSet s)
-NO permiten duplicados, sobrescribir los métodos equals() y hasCode()
HashMap
HashMap()
HashMap(int initialCapacity)
HashMap(int initialCapacity, float loadFactor)
HashMap(Map m)
TreeMap
TreeMap()
TreeMap(Comparator c)
TreeMap(Map m)
TreeMap(SortedMap m)
-asociaciones clave/valor, NO permite claves duplicados
interfazes Comparable y Compararator
int compareTo(Object o)
int compare(Object o1, Object o2)
-recomendable consistente con equals()
-devuelve
un int <0 si menor
un int =0 si igual
un int >0 si mayor
Vector, Stack y Hashtable
-implementaciones heredadas de la versión 1.1
-todos sus métodos están sincronizados
-> implica rendimiento muchísimo menor
-para sincronizar
-> utilizar Collections.sincronizedX()
initialCapacity (capacidad inicial)
capacity (capacidad)
size (tamaño)
loadFactor (factor de carga)
-objetos de tipo Collection o Map son contenedores
a diferencia de las matrices,
incrementan su capacidad cuando lo necesitan
-loadFactor = size / capacity
-si size > loadFactor
-> se incrementa la capacidad
-> se crea una nueva estructura de datos
-> se copia los elementos de una a otra
-si contenedor es de tipo hash
-> implica distribución uniforme de los elementos
-para optimizar
-> para evitar amplicaiones sucesivas
-> initialCapacity lo más cercano al tamaño esperado
public int hashCode()
-se utiliza como indice
-sobrescribir el método equals()
-> implica sobrescribir el método hashCode()
-a.equals(b) == true
-> implica a.hashCode() == b.hashCode()
-pero a.hashCode() == b.hashCode()
-> NO implica a.equals(b) == true
→ pero augmenta velocidad
-su cálculo ha de ser rápido
-los valores devueltos deben de ser uniformemente distribuidos en toda la tabla
public boolean equals()
-reflexiva a.equals(a) == true
-simétrica a.equals(b) == b.equals(a)
-transitiva a.equals(b) == b.equals(c) == true
-> implica a.equals(c) == true
-a == b
-> implica a.equals(b) == true
-b == null
-> implica a.equals(b) == false
links
http://www.javahispano.org/download.d
http://www.javahispano.org/download.d
INTERFACES
IMPLEMENTACIONES
Autor: Adolfo Sanz De Diego asanzdiego@yahoo.es