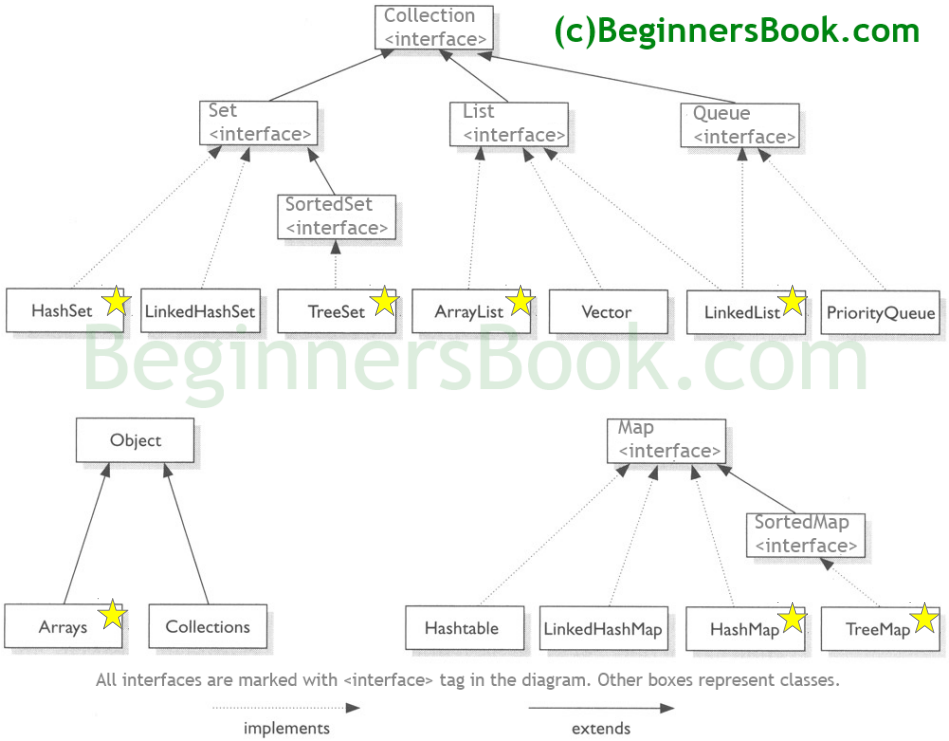
PROGRAMACIÓN ORIENTADA A OBJETOS

TEMA4: Más Collections: Conjuntos y Mapas

Manel Guerrero

PROGRAMACIÓN ORIENTADA A OBJETOS

Antes que nada veamos W4H1
Para repasar Comparable.



CONTAINERS (o COLLECTIONS)

```
java.lang.Object
+---java.util.AbstractCollection<E>
|   +---java.util.AbstractList<E>
|   |   +---ArrayList (*)
|   |   +---AbstractSequentialList
|   |       +---LinkedList (*)
|   +---java.util.AbstractSet<E>
|   |   +---java.util.HashSet<E> (*)
|   |   +---java.util.TreeSet<E> (*)
|   +---java.util.AbstractQueue<E>
|       +---...
+---java.util.AbstractMap<K,V>
    +---java.util.HashMap<K,V> (*)
    +---java.util.TreeMap<K,V> (*)
```

- La imagen anterior o es de una versión anterior o contiene errores.
- Esta es la versión correcta de la JavaSE8.
- En el tema anterior vimos las listas. Ahora veremos los conjuntos (sets) y los diccionarios o mapas (maps).

Conjuntos (Sets)

- No hay un orden (como en las listas).
- No pueden haber elementos duplicados.
- Utilizaremos HashSet cuando los elementos no tienen un orden.
- TreeSet cuando los elementos son comparables (implementan la interface Comparable).
- Si nos da igual, HashSet es más eficiente.
- Veamos ahora W4H2.

Diccionarios (Maps)

- En lugar de guardar elementos <E>, guardan parejas de clave y valor <K,V>.
- Como en un diccionario donde tenemos <palabra,definición>.
- Las claves no pueden estar repetidas. Tienen que ser únicas.
- Se pueden usar cuando un atributo de V es único.
  - Ejemplo: <NIF,persona>.
- Igual que con los Sets, utilizaremos HashSet cuando no necesitemos ordenar los elementos.
- TreeSet cuando necesitemos ordenarlos.
- Veamos ahora W4H3 y W4H4.

Iterando por una lista

```
Iterator<Book> it = llista.iterator();
while(it.hasNext()){
    Book b = it.next();
    if(b.GetIsbn().equals(my_book){
        System.out.println(b);
        break;
    }
}
```

Iterando por una lista (casting requerido)

```
// En el ejemplo anterior:
// Iterator<Book> it = llista.iterator();
Iterator it = llista.iterator();
while(it.hasNext()){
    Book b = (Book) it.next();
    if(b.GetIsbn().equals(my_book)){
        System.out.println(b);
        break;
    }
}
```

- Aquí hace falta un casting porque no hemos indicado que el iterador era de Books.
- En el ejemplo anterior habíamos declarado el iterador como “Iterator<Book>”.

Enhanced for for collections

- For lists and sets:

  for(Book b : set)

- For maps:

  for(Book b : map.values())

  for(String isbn : map.keys())

  for(Map.Entry<String, Book> entry : map.entrySet())

- El for-each en un mapa tanto se puede hacer:
  - Por valor.
  - Por clave.
  - Por “entry” (pareja <clave,valor>).

Enhanced for for collections (2)

- Si iteras por entry puedes acceder tanto a la clave como al valor:

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>();
for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
        System.out.println("Key = " + entry.getKey() +
          ", Value = " + entry.getValue());
}
```

- Pero si solo quieres las claves o el valor:

```
for (Integer k : map.keySet())
for (Integer v : map.values())
```

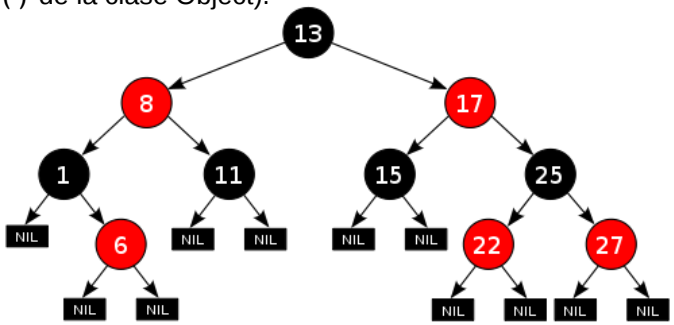
TreeSet, TreeMap y Comparable

```
To use TreeSet<Dog>, Dog (<E>) has to be Comparable.
To use TreeMap<Key,Dog>, Key (<K>) has to be Comparable.

class Dog implements Comparable<Dog>{
  String color;
  int size;
  [...]
  @Override
  public int compareTo(Dog o) {
    return  o.size - this.size;
  }
}
```

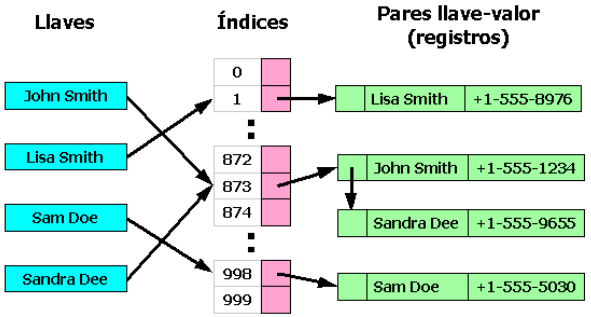
Tree: Árboles rojo-negro

- Los TreeSet y TreeMap están implementados como árboles rojo-negro: un tipo de árbol binario de búsqueda.
- Por tanto las operaciones de acceso, inserción y borrado son complicadas, pero sus tiempos de ejecución son O(log n). Y sus elementos están siempre ordenados.
- Para ordenarlas usa el `compareTo()` (por tanto hay que implementar la interface Comparable) y el `equals()` (por tanto hay que override el `equals()` de la clase Object).

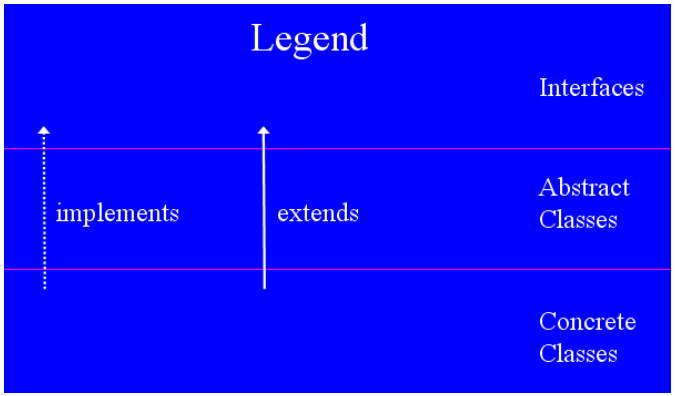


Hash: Tablas de Hash

- Los HashSet y HashMap están implementados como tablas de hash.
- Por tanto las operaciones de acceso, inserción y borrado tienen tiempos de ejecución constante si la función de hash distribuye correctamente. Y sus elementos no están ordenados.
- En una clase podremos override el "`public int hashCode()`" de la clase Object. Pero normalmente no será necesario.

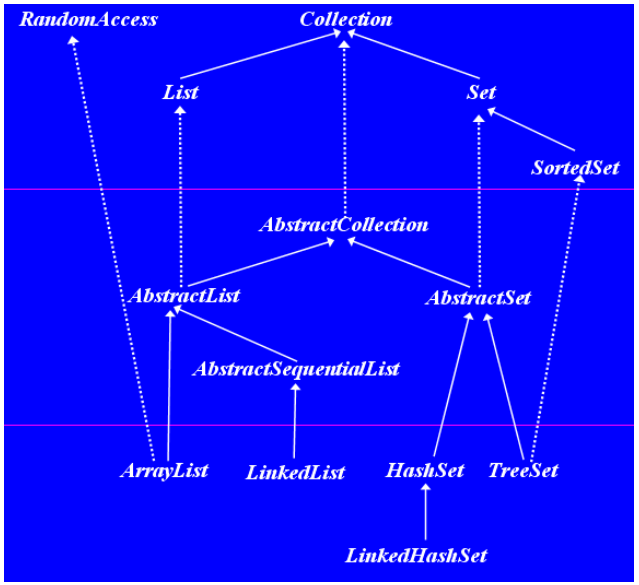


Interfaces, abstract and concrete classes of collections

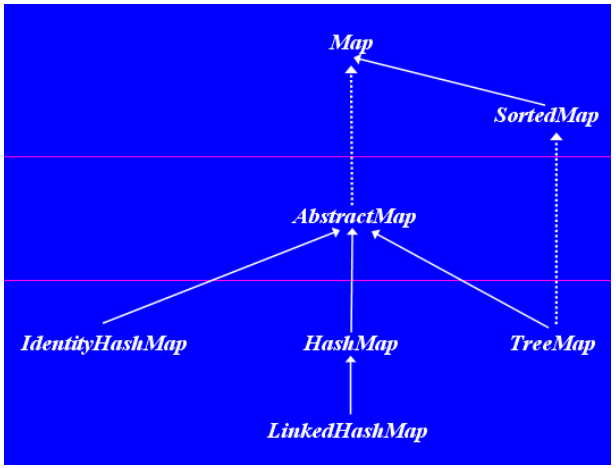


https://www.cs.cmu.edu/~pattis/15-1XX/15-200/lectures/collectionsiii/lecture.html

Interfaces, abstract and concrete classes of collections (2)



Interfaces, abstract and concrete classes of collections (3)



The difference between an Interface and an Abstract class

- **Methods of a Java interface are implicitly abstract and cannot have implementations. A Java abstract class can have instance methods that implements a default behaviour. [No del todo cierto desde Java8]**
- Variables declared in a Java interface are by default final.
- Members of a Java interface are public by default.
- A Java interface should be implemented using keyword “implements”; A Java abstract class should be extended using keyword “extends”.
- **A Java class can implement multiple interfaces but it can extend only one abstract class.**

  http://stackoverflow.com/questions/18777989/how-should-i-have-explained-the-difference-between-an-interface-and-an-abstract

The difference between an Interface and an Abstract class (2)

- An **interface** is a description of the behaviour an implementing class will have. The implementing class ensures, that it will have these methods that can be used on it. It is basically a contract or a promise the class has to make.
- An **abstract class** is a basis for different subclasses that share behaviour which does not need to be repeatedly created. Subclasses must complete the behaviour and have the option to override predefined behaviour (as long as it is not defined as final or private).
- Unrelated classes can have [same] capabilities through interface but related classes change the behaviour through extension of base classes.

The difference between an Interface and an Abstract class (3)

- You will find good examples in the java.util package which includes interfaces like List and abstract classes like AbstractList which already implements the interface. The official documentation describes the AbstractList as follows:

  “This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a "random access" data store (such as an array).

The difference between an Interface and an Abstract class (4)

- An interface consists of singleton variables (public static final) and public abstract methods. We normally prefer to use an **interface** in real time **when we know what to do but don't know how to do it**.
- We choose an **abstract class when there are some features for which we know what to do, and other features that we know how to [do it]**.

PROGRAMACIÓN ORIENTADA A OBJETOS

Preguntas