Algoritmos y Estructuras de Datos

Universidad Católica del Uruguay

Diccionarios, Mapas y API de Colecciones

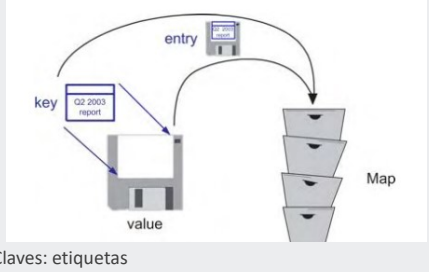
1

Mapas

- Un “mapa” nos permite almacenar elementos de manera tal que puedan ser rápidamente ubicados utilizando claves.
- Cada elemento contiene valiosa información adicional a su clave
- Un mapa almacena pares ***clave-valor*** ***(k,v)***, tales que cada clave es única

2

Mapas



- Claves: etiquetas
- Valores (discos)
- Entrada en el mapa(caja): disco etiquetado
- Las claves pueden ser usadas luego para buscar y recuperar los discos

3

TDA Mapa

- Almacena una colección de objetos en la forma ***clave-valor***
  - *tamaño()*
  - *estaVacio()*
  - *recuperar(k)*
  - *poner(k,v)*
  - *eliminar(k)*
  - *claves()*
  - *valores()*
  - *elementos()*

4

Diccionarios

- Como los mapas, también almacenan pares ***clave-valor***
- El diccionario permite múltiples entradas con la misma clave
- Dos tipos:
  - Ordenados
  - desordenados

5

TDA Diccionario

- *tamaño()*
- *estaVacio()*
- *buscar(k)*
- *buscarTodos(k)*
- *insertar(k,v)*
- *eliminar(e)*
- *elementos()*
  - *Cada elemento tiene **getKey** y **getValue***

6

API Colecciones JAVA

- Una *colección* – a veces llamada *contenedor* - es un objeto que agrupa múltiples elementos en una sóla unidad.
- Las colecciones se utilizan para almacenar, recuperar, manipular y comunicar los datos agregados.
- El framework de colecciones:
  - Interfases
  - Implementaciones
  - algoritmos

7

¿Cuál de los siguientes ***no*** es un beneficio de usar la librería de colecciones?

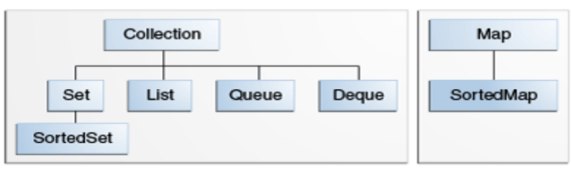
a) Reducir el esfuerzo de programación
b) Incrementar la velocidad y calidad del programa
c) Reducir el esfuerzo para aprender y usar nuevas APIs (muchas usan las colecciones como entrada y / o salida)
d) Promover la reutilización del software
e) Ahorrar en el desarrollo de casos de prueba

8

API Colecciones – implementaciones(parcial)

- Set
  - HashSet
  - TreeSet
  - LinkedHashSet
- List
  - ArrayList
  - LinkedList
- Map
  - HashMap
  - TreeMap
  - LinkedHashMap

9



The core collection interfaces.

10

API Colecciones - Interfaz "Collection"

- int size(),
- boolean isEmpty(),
- boolean contains(Object element),
- boolean add(E element),
- boolean remove(Object element),
- Iterator<E> iterator().
- Recorridos:
  - for-each
  - Iterador
  - Operaciones agregadas
- "bulk operations"
- Array operations

11

API Colecciones - Interfaz "Set"

- Colección que no puede tener elementos duplicados
- Modela la abstracción matemática "conjunto"
- Sólo contiene métodos heredados de Collection.
- Implementaciones
  - HashSet
  - TreeSet
  - LinkedHashSet

12

API Colecciones - Interfaz “Set”

- Supongamos que tenemos una Collection, “palabras”, y que deseamos obtener otra en la que no hayan duplicados, “palabras_sin_dups”…
- ¿cuál sería una forma rápida de lograrlo? Justifícalo….
- ¿cuáles son las operaciones “bulk” definidas en la interfaz “set”?

13

API Colecciones - Interfaz “Set” usando “bulk operations”:

1. ¿cómo implementar la operación “unión” de conjuntos?
2. ¿cómo implementar la operación “intersección” de conjuntos?
3. ¿cómo implementar la operación “diferencia” de conjuntos?

14

API Colecciones - Interfaz “List”

- Es una Colección ordenada (secuencia)
- Puede contener elementos duplicados
- Operaciones adicionales (a las de Collection)
  - Acceso por posición (get, set, add, addAll, remove)
  - Búsqueda (indexOf, lastIndexOf)
  - Iteración (extiende la semántica de Iterator – métodos de listIterator
  - Vista de subrango (método sublist)
- Dos implementaciones:
  - ArrayList
  - LinkedList

15

API Colecciones - Interfaz “List”

- ¿qué funcionalidades ofrece el listIterator?

1. hasNext
2. next
3. remove
4. hasPrevious
5. previous

- ¿cómo se usa? Ejemplo….

16

API Colecciones - Interfaz “List”

- Vista de sub-rangos
- *subList(int fromIndex, int toIndex)*
  - *for (int i = fromIndex; i < toIndex; i++) { … }*
- Elimina la necesidad de operaciones de subrangos explícitas …
  - *Ej: list.subList(fromIndex, toIndex).clear();*

17

Algoritmos de Listas

- La mayoría de los algoritmos polifmórficos de la clase Collections se aplican específicamente a List:
  - sort — ¿cómo funciona? ¿es “estable”?
  - shuffle — ¿qué hace?
  - reverse.
  - rotate — ¿qué hace?
  - swap —.
  - replaceAll
  - fill
  - copy.
  - binarySearch
  - indexOfSubList
  - lastIndexOfSubList

18

API Colecciones - Interfaz “Map”

- Un Map mapea claves a valores.
- Cada clave puede mapear como máximo a un valor
- Operaciones básicas
  - put
  - get
  - remove
  - containsKey
  - containsValue
  - size
  - empty
- Operaciones “ bulk”
  - putAll
  - Clear
- Vistas
  - keySet, entrySet, values

19

API Colecciones - Interfaz “Map” implementaciones

- HashMap
- TreeMap
- LinkedHashMap

20

Collection views

- Los métodos de vistas de *Collection* permiten que un *Map* sea visto como una *Collection* de tres formas:
  - keySet
  - values
  - entrySet

21

Collection views

- Las vistas de *Collection* proveen la ***única forma*** de iterar sobre un ***Map***.

```
for (KeyType key : m.keySet())
    System.out.println(key);
```

- y con un iterador:

```
for (Iterator<Type> it = m.keySet().iterator(); it.hasNext(); )
        if (it.next().isBogus()) it.remove();
```

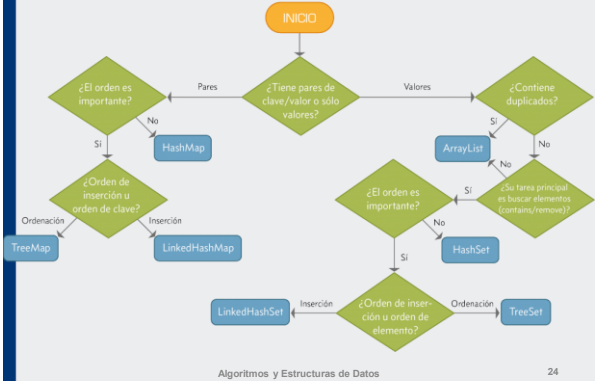

22

Criterios para elegir la implementación adecuada

- ¿Qué se almacenará en la colección?
  - Objetos (solo valores) o pares clave / valor?
  - Se aceptan datos duplicados?
- Performance ¿Qué acciones se realizarán sobre la colección? ¿Cuál es el cometido?
  - Se utilizará para búsqueda o se realizarán constantes inserciones y eliminaciones?
  - Importa el orden de inserción?
  - Es necesario tener los datos ordenados?
- ¿Se trabaja en un ambiente de concurrencia?

23

Diagrama de decisión para uso de colecciones Java



24

Interfaces	Implementaciones				
	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

25

Property	HashMap	LinkedHashMap	TreeMap
Time Complexity (Big O notation) Get, Put, ContainsKey and Remove method	O(1)	O(1)	O(log n)
Iteration Order	Random	Sorted according to either Insertion Order or Access Order (as specified during construction)	Sorted according to either natural order of keys or comparator (as specified during construction)
Null Keys	allowed	allowed	Not allowed if Key uses Natural Ordering or Comparator does not support comparison on null Keys
Interface	Map	Map	Map, SortedMap and NavigableMap
Synchronization	none, use Collections.synchronizedMap()	None, use Collections.synchronizedMap()	none, use Collections.synchronizedMap()
Data Structure	List of Buckets, if more than 8 entries in bucket then Java 8 will switch to balanced tree from linked list	Doubly Linked List of Buckets	Red-Black Tree (a kind of self-balancing binary search tree) implementation of Binary Tree. This data structure offers O (log n) for insert, Delete and Search operations and O (n) Space Complexity
Applications	General Purpose, fast retrieval, non-synchronized. ConcurrentHashMap can be used where concurrency is involved.	Can be used for LRU cache, other places where insertion or access order matters	Algorithms where Sorted or Navigable features are required. For example, find among the list of employees whose salary is next to given employee, Range Search, etc.
Requirements for Keys	Equals() and hashCode() needs to be overwritten	Equals() and hashCode() needs to be overwritten	Comparator needs to be supplied for Key implementation, otherwise natural order will be used to sort the keys

26

Uso de hashMap

- ¿cómo es la estructura interna del hashMap? (dibújala!!!!)
- ¿cuáles son los parámetros que tiene esta colección?
- ¿cuáles son los valores por defecto?

27

Uso de hashMap

- En el TA3 utilizamos un conjunto de palabras “listado-general_desordenado.txt” para probar los tiempos de inserción en distintas estructuras.
- Genera un nuevo programa para medir los tiempos para inserción en hashMap, con 3 diferentes valores de los parámetros (primero usa los valores por defecto)
- Obtén los resultados de tiempos en la planilla, normaliza y grafica.

28

Collection	Ordering	Random Access	Key-Value	Duplicate Elements	Null Element	Thread Safety
ArrayList	✓	✓	✗	✓	✓	✗
LinkedList	✓	✗	✗	✓	✓	✗
HashSet	✗	✗	✗	✗	✓	✗
TreeSet	✓	✗	✗	✗	✗	✗
HashMap	✗	✓	✓	✗	✓	✗
TreeMap	✓	✓	✓	✗	✗	✗
Vector	✓	✓	✗	✓	✓	✓
Hashtable	✗	✓	✓	✗	✗	✓
Properties	✗	✓	✓	✗	✗	✓
Stack	✓	✗	✗	✓	✓	✓
CopyOnWriteArrayList	✓	✓	✗	✓	✓	✓
ConcurrentHashMap	✗	✓	✓	✗	✗	✓
CopyOnWriteArraySet	✗	✗	✗	✗	✓	✓

29

Contrato entre hashCode() y equals()

- Cada vez que se invoca en el mismo objeto más de una vez durante la ejecución de una aplicación Java, **hashCode()** debe devolver el mismo entero, siempre que no se modifique la información utilizada en las comparaciones de igualdad en el objeto.
- Este entero no necesita permanecer consistente entre dos ejecuciones de la misma aplicación o programa.
- Si dos objetos son iguales de acuerdo con el método **equals()**, entonces llamar a **hashCode()** en cada uno de los dos objetos debe producir el mismo resultado entero.
- No se requiere que si dos objetos son desiguales de acuerdo con **equals()**, entonces llamar a **hashCode()** en cada uno de los dos objetos debe producir resultados enteros distintos.
- Java hashCode() and equals() Methods - HowToDoInJava

30

hashCode()

- Indica en un documento cuáles son las implementaciones estándar de hashCode() para los siguientes tipos:
  - Int
  - String
  - Float
  - Object
- ¿cómo definirías el hashCode() para números de cédula, sabiendo que los que vas a considerar son uruguayos de entre 20 y 25 años de edad?

31

hashCode() vs equals()

- ¿cuál es la relación entre hashCode y equals?
- ¿cómo funcionan estos métodos cuando se inserta o busca en un hashMap?
  - Dibuja la estructura interna del hashMap de Java
- ¿Cuáles son los parámetros posibles para el constructor del hashMap?
  - ¿cómo los seleccionarías, de acuerdo al problema, tiempo de ejecución, etc.?

32

Revisión PD7

Dada la siguiente clase:

```
public class User {

    private long id;
    private String name;
    private String email;

    // standard getters/setters/constructors
}
```


33

¿Es correcta la siguiente implementación?

```java
public class User {
  // ...

  @Override
  public int hashCode() {
      return 1
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null) return false;
    if (this.getClass() != o.getClass()) return false;
    User user = (User) o;
    return id == user.id
        && (name.equals(user.name)
        && email.equals(user.email));
    }
}
```

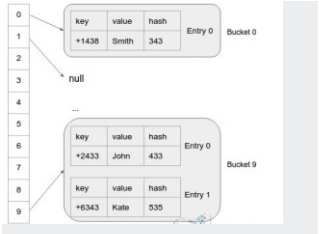

34

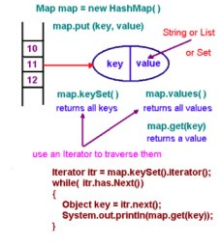
¿Es correcta la siguiente implementación?

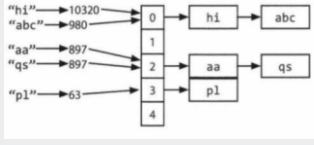
```java
public class User {
  // ...

  @Override
  public int hashCode() {
    int hash = 7;
    hash = 31 * hash + (int) id;
    hash = 31 * hash + (name == null ? 0 : name.hashCode());
    hash = 31 * hash + (email == null ? 0 : email.hashCode());
    return hash;
}
```


35







36

Links útiles sobre hashCode en Java

- https://www.baeldung.com/java-hashcode
- https://es.wikipedia.org/wiki/HashCode()_(Java)
- https://www.cs.cmu.edu/~adamchik/15-121/lectures/Hashing/hashing.html
- String Hashing - Algorithms for Competitive Programming (cp-algorithms.com)
- https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html