Unió Europea
Fons Social Europeu
L'FSE inverteix en el teu futur

GENERALITAT VALENCIANA

I.E.S. Maria Enríquez
Gandia

UF11 - JAVA COLLECTIONS FRAMEWORK

- Teoria -

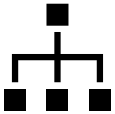
PROGRAMACIÓ
CFGS DAM

Autor:
Àngel Olmos Giner
a.olmosginer@edu.gva.es
2022/2023

JAVA COLLECTIONS FRAMEWORK

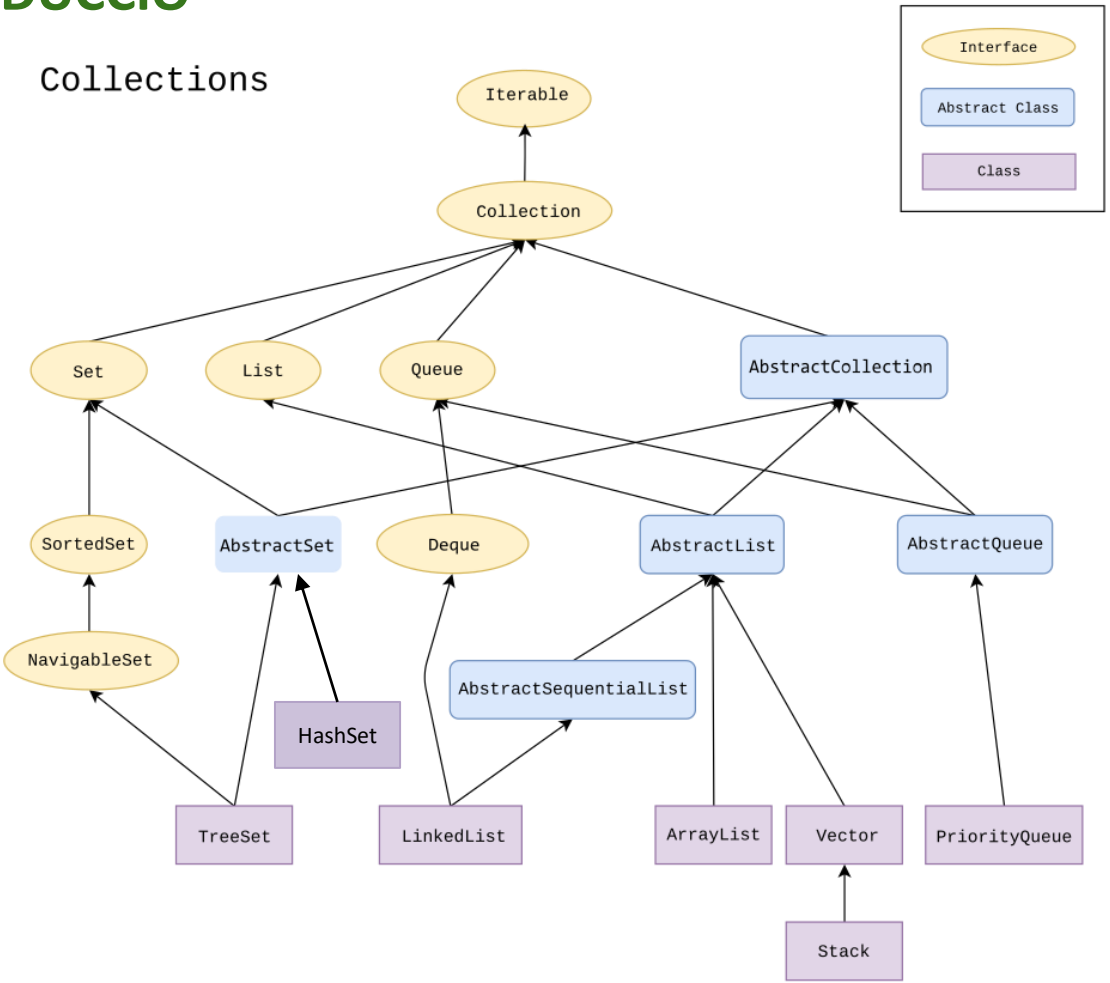
ÍNDEX DE CONTINGUTS

1. INTRODUCCIÓ

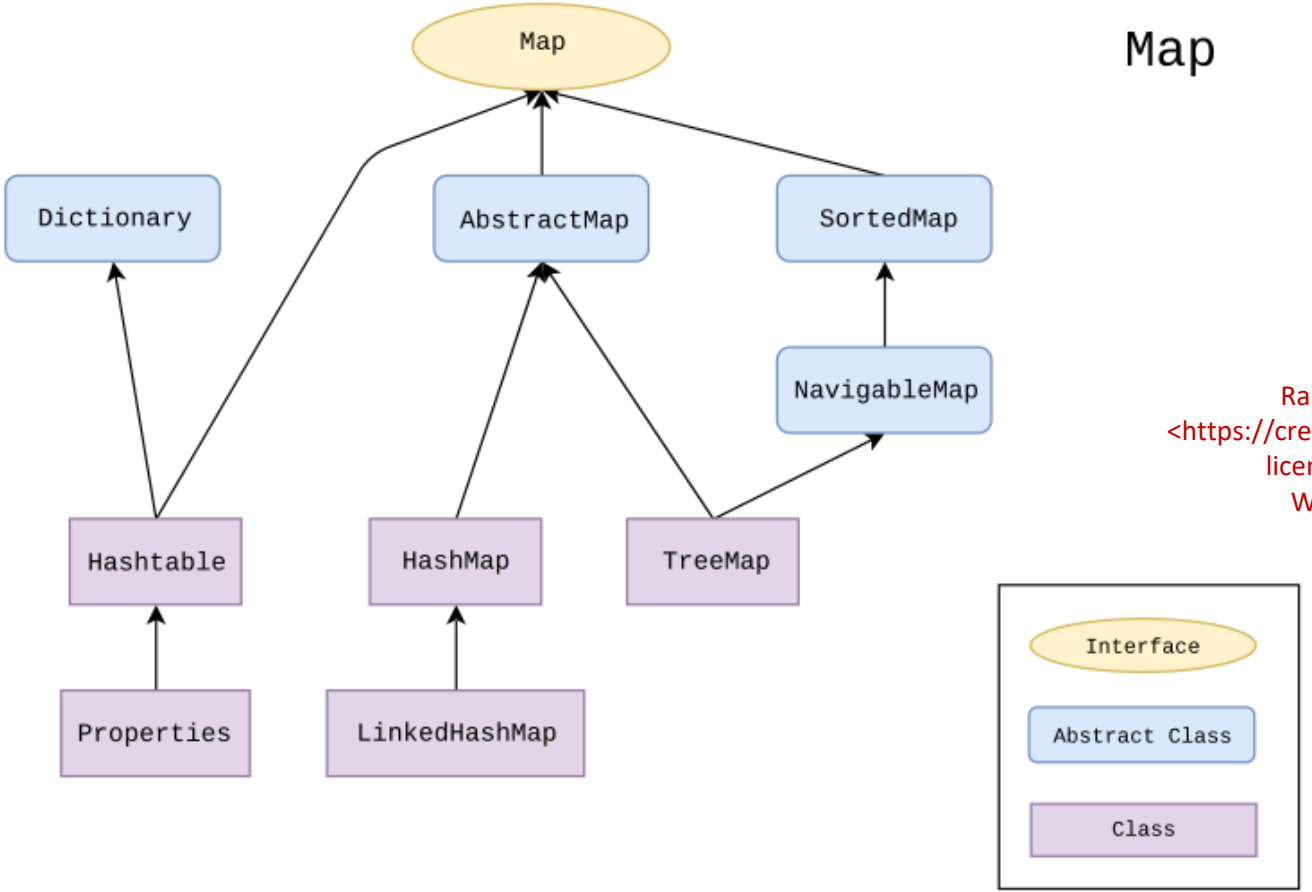


- Una **col·lecció és un objecte que representa un grup d'objectes** (p.e. la classe *ArrayList* que ja conegueu)
- El *Collections Framework* és una arquitectura unificada per representar i manipular col·leccions → un **conjunt de classes i interfícies que implementen estructures de dades de col·lecció**
- Les col·leccions i els *arrays* són similars perquè totes dues contenen referències a objectes, però a diferència dels *arrays*, les col·leccions no cal que se'ls assigne una certa capacitat quan s'instancien i **poden créixer i reduir-se automàticament** quan s'afegeixen o s'eliminen
- Les col·leccions **no poden contenir tipus primitius** (int, double ...) **però** en canvi **poden contenir *Wrapper Classes*** com Integer, Double ...
- **Les interfícies** del *Collection Framework* **es divideixen en dos grups**. Les basades en ***java.util.Collection*** **i** les basades en ***java.util.Map***

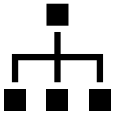
1. INTRODUCCIÓ



1. INTRODUCCIÓ



1. INTRODUCCIÓ



Les implementacions de propòsit general es resumeixen a la següent taula

General purpose implementations

Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue, Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

https://docs.oracle.com/en/java/javase/19/docs/api//java.base/java/util/doc-files/coll-overview.html

Visita l'enllaç i anem a vorer les característiques d'algunes de les classes
És important que entengues el que es mostra a la documentació oficial de Java i que et sàpies moure pels diferents apartats

JAVA COLLECTIONS FRAMEWORK

ÍNDEX DE CONTINGUTS

2. IMPLEMENTACIONS

Interfície *List*

General purpose implementations

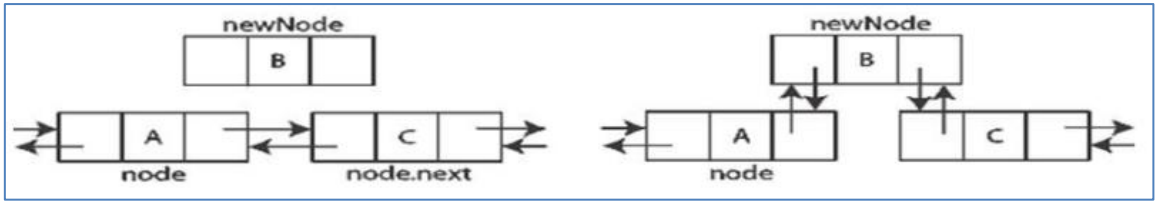
Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue, Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

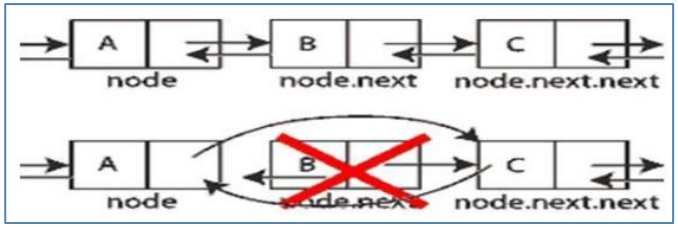
- Els elements tenen un ordre específic, es permeten elements duplicats i *null*
- Els elements es poden col·locar en una posició específica i es poden buscar dins de la llista
- Exemples de classes concretes que implementen *List* són:
  - *java.util.ArrayList*: és la implementació d'un *array* redimensionable
  - *java.util.LinkedList*: és la implementació de la llista doblement enllaçada

2. IMPLEMENTACIONS

Interfície *List*

- *ArrayList* i *LinkedList* tenen diferències en el seu rendiment a causa del tipus d'implementació
- *LinkedList* és ràpid per a agregar/eliminar elements però és lent si accedim a un element específic (accés seqüencial, des del principi/final de la llista)





- *ArrayList* és ràpid per a accedir a un element específic (*random access*) però lent si agreguem/eliminem elements (sobretot pel mig → requereix desplaçar molts elements)

JAVA COLLECTIONS FRAMEWORK

ÍNDEX DE CONTINGUTS

2. IMPLEMENTACIONS

Interfície *Set*

General purpose implementations

Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue, Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

- Un *Set* no pot tindre cap element duplicat
- Els elements no es poden trobar per índex
- Exemples de classes concretes que implementen *Set* són:
  - *java.util.HashSet*: utilitza *hash codes* per evitar duplicats i no té un ordre establert
  - *java.util.TreeSet*: utilitza una estructura de dades *red–black tree* (arbres binaris) que s'assegura que no hi hagi duplicats i ordena els elements de forma natural

2. IMPLEMENTACIONS

Interfície *Set*

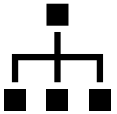
Parameters	HashSet	TreeSet
Data Structure	Hash Table	Red-black Tree
Time Complexity (add/remove/contains)	O(1)	O(log n)
Iteration Order	Arbitrary	Sorted
Null Values	Allowed 1	Not Allowed
Processing	Fast	Slow

- El rendiment de *TreeSet* és lent en comparació amb *HashSet* perquè ordena els elements després de cada operació d'inserció o esborrat
- Utilitzeu *TreeSet* si voleu que els elements estiguen ordenats, sinó feu servir *HashSet* perquè el seu rendiment és més ràpid

https://www.javatpoint.com/hashset-vs-treeset-java

2. IMPLEMENTACIONS

Interfície *Set*



```
public static void main(String[] args) {
//creating a Set
    Set<String> s = new TreeSet<String>();
//adding elements
    s.add("Dell");
    s.add("HP");
    s.add("Apple");
    s.add("Acer");
    s.add("Asus");
    s.add("Samsung");
//try to add a duplicate element
    s.add("Asus");
//prints the Set elements on the console
    System.out.println("Set elements are: ");
    for (String temp : s) {
        System.out.println(temp);
    }

    s.add("Lenovo");

    System.out.println("Set elements are: ");
    for (String temp : s) {
        System.out.println(temp);
    }
}
```

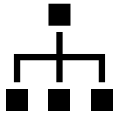
Implementa aquest codi, després canvia a *HashSet* i observa les diferències

JAVA COLLECTIONS FRAMEWORK

ÍNDEX DE CONTINGUTS

2. IMPLEMENTACIONS

Interfície *Queue / Deque*



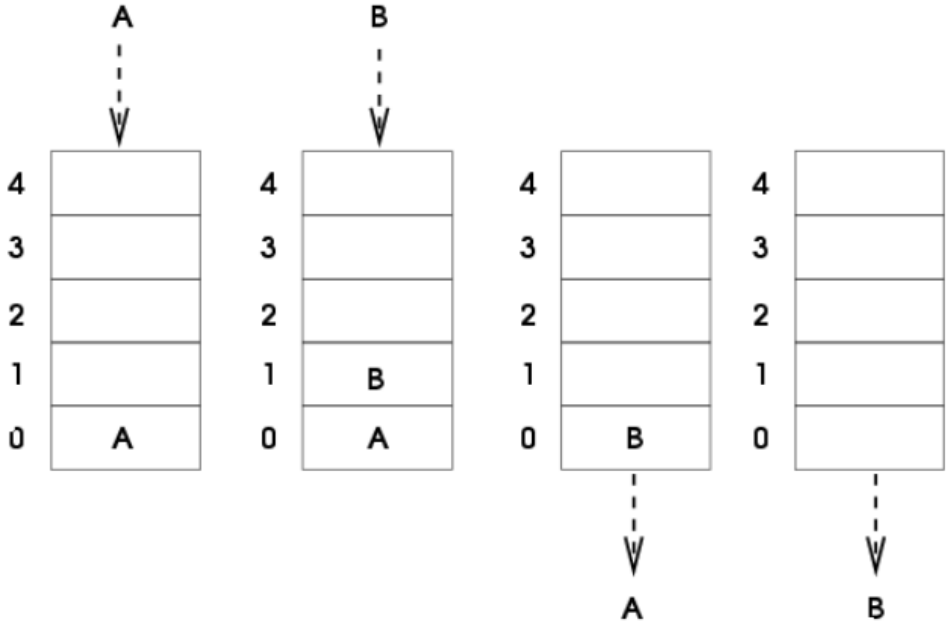
General purpose implementations

Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue, Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

- Una cua (*Queue*) és una llista basada en el criteri FIFO (*first in / first out* → el primer que entra és el primer que ix)
- Els nous elements s'afigen en la part de darrere i l'extracció d'elements es realitza per davant
- Exemples de classes concretes que implementen *Queue* són:
  - *java.util.LinkedList*: és la implementació amb llista doblement enllaçada
  - *java.util.ArrayDeque*: és la implementació amb *array* redimensionable

2. IMPLEMENTACIONS

Interfície *Queue / Deque*



Aquesta estructura s'inspira en solucions de la vida real ... saps algun exemple?

- ✓ la cua del supermercat
- ✓ la cua de treballs en una impressora
- ✓ etc ...

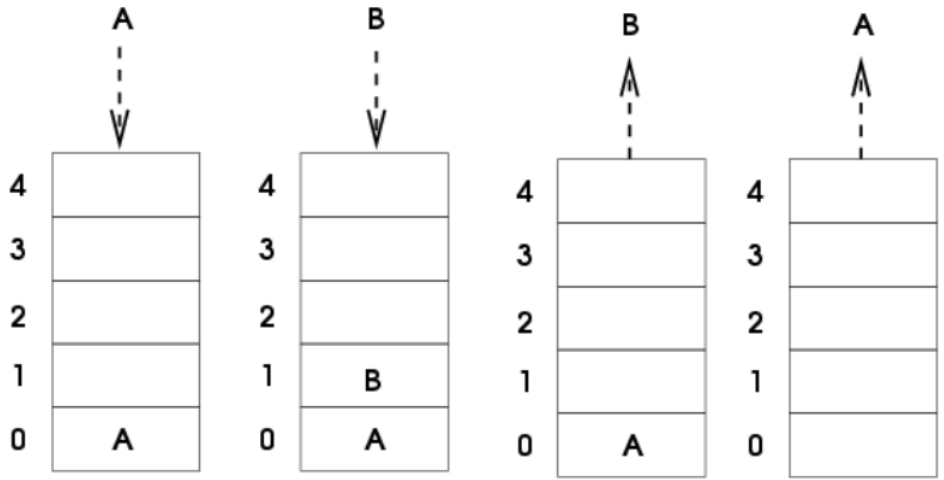
JAVA COLLECTIONS FRAMEWORK

ÍNDEX DE CONTINGUTS

2. IMPLEMENTACIONS

Classe *Stack*

- Una pila (*Stack*) és una llista basada en el criteri LIFO (last in / first out, l'últim que entra és el primer que ix)



Símils en la vida real ??

- ✓ una pila de llibres sobre una taula
- ✓ pila de papers que usa la impressora
- ✓ Gestió de productes peribles (*perecederos*) d'una tenda: Els productes amb data de caducitat més recents es venen primer

JAVA COLLECTIONS FRAMEWORK

ÍNDEX DE CONTINGUTS

2. IMPLEMENTACIONS

Interfície *Map*

General purpose implementations

Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue, Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

- Els mapes (*Maps*) són estructures de dades simples que associen una clau amb un element
- Això permet que el mapa sigui molt flexible
- Exemples de classes concretes que implementen *Maps* són:
  - *java.util.HashMap*: és la implementació utilitzant taules *Hash*
  - *java.util.TreeMap*: és la implementació utilitzant arbres binaris

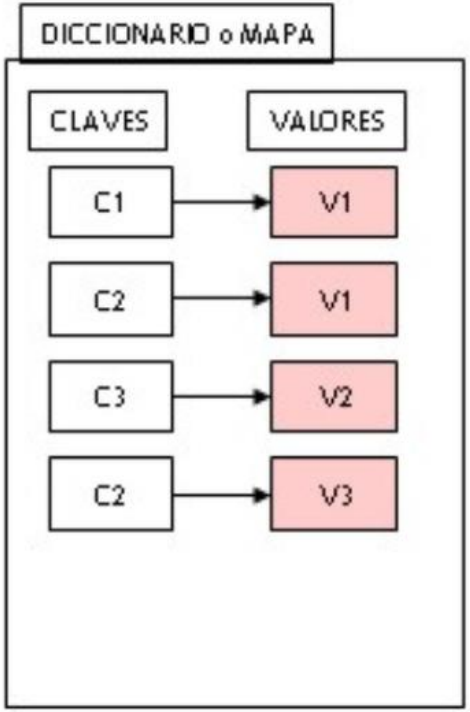
2. IMPLEMENTACIONS

Interfície *Map*

- Fins ara totes les estructures de dades que hem vist emmagatzemen directament la dada en qüestió, i per a recuperar-la necessitem el seu índex o una còpia de la dada

- A vegades en el món real ens trobem amb situacions en les quals usem algun tipus de "clau" per a recuperar les dades

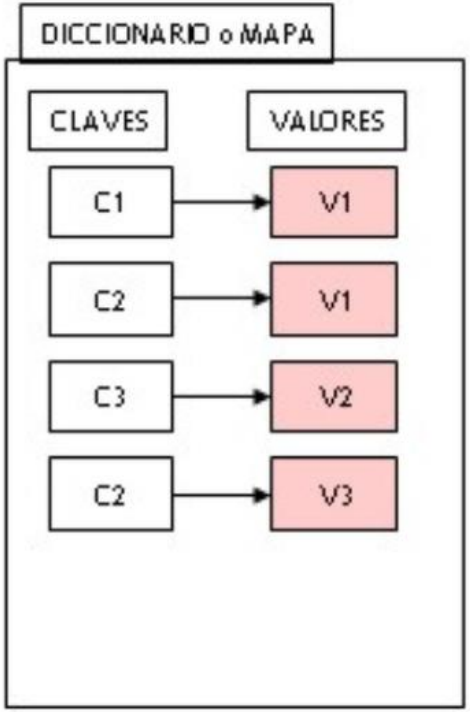
- Un diccionari o mapa és una estructura dinàmica de dades formada per una sèrie de parelles “clau → valor”



2. IMPLEMENTACIONS

Interfície *Map*

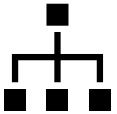
- En un mapa, els diferents elements poden trobar-se ordenats o no segons la seua clau
- Depenent del tipus de mapa és possible que es permeten dades repetides o que no es permeta la repetició de dades
- Coneixent la clau és possible recuperar el valor corresponent ... sabeu exemples vida real?
    - una guia telefònica: clau → nom persona, valor → telèfon
    - un diccionari: clau → paraula, valor → definició
    - una targeta bancària
    - dades de login ...



JAVA COLLECTIONS FRAMEWORK

ÍNDEX DE CONTINGUTS

3. LA CLASSE *HASHMAP*

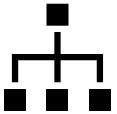


Alguns dels mètodes més importants de la classe *HasMap* són:

- **get(clau):** Obté el valor corresponent a una clau. Retorna *null* si no existeix
- **put(clau, valor):** Afig un parell (clau, valor) al diccionari. Si ja hi havia un valor per a aqueixa clau, es sobreescriu
- **keySet():** Retorna un conjunt (*set*) amb totes les claus
- **values():** Retorna una col·lecció amb tots els valors
- **entrySet():** Retorna una col·lecció amb tots els parells (clau, valor)
- **containsKey(clau)**: Retorna *true* si el diccionari conté la clau indicada

```
HashMap<TipusClau, TipusValor> mapa = new HashMap< TipusClau, TipusValor >()
```

3. LA CLASSE *HASHMAP*



```
public static void main(String[] args) {
    Map<Integer, String> m = new HashMap<>();
    m.put(924, "Amalia Núñez");
    m.put(921, "Cindy Nero");
    m.put(700, "César Vázquez");
    m.put(219, "Víctor Tilla");
    m.put(537, "Alan Brito");
    m.put(605, "Esteban Quito ");
    System.out.println("Els elements de m són: \n" + m);
}
```

Implementa aquest codi, després canvia a *TreeMap* i observa les diferències

3. LA CLASSE *HASHMAP*

Afegix aquest codi a la versió amb *HashMap* i observa el resultat

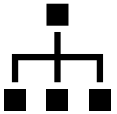
```
System.out.println(m.get(921));
System.out.println(m.get(605));
System.out.println(m.get(888)); // aquesta clau no existeix

System.out.println("\nTotes les entrades del diccionari extretes amb entrySet:");
System.out.println(m.entrySet());
```

Fes els canvis necessaris per a que l'eixida siga com aquesta ...

```
Entrades del diccionari extretes una a una:
921=Cindy Nero
537=Alan Brito
219=Víctor Tilla
924=Amalia Núñez
700=César Vázquez
605=Esteban Quito
```

... i la versió utilitzant una expressió *Lambda*?

3. LA CLASSE *HASHMAP*

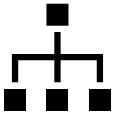


Implementa un mètode anomentat “mostrarMap” que mostre per pantalla el contingut del diccionari de la següent forma. Busca quins mètodes de HashMap et poden ser útils

Codi	Nom
921	Cindy Nero
537	Alan Brito
219	Víctor Tilla
924	Amalia Núñez
700	César Vázquez
605	Esteban Quito

... i la versió “mostrarMapLambda” utilitzant una expressió *Lambda*?

3. LA CLASSE *HASHMAP*

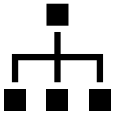


Modifica el programa per a demanar a l'usuari un codi i mostrar el seu valor per pantalla. Si el codi no existeix, cal mostrar un missatge d'error

```
Per favor, introdueix un codi: 700
El codi 700 correspon a César Vázquez
```

```
Per favor, introdueix un codi: 666
El codi no existeix.
```

3. LA CLASSE *HASHMAP*



Per últim, esborra un registre existent al diccionari i mostra el resultat

```
### Esborrrar entrada 700 del diccionari

Codi     Nom
------   ------------
921      Cindy Nero
537      Alan Brito
219      Víctor Tilla
924      Amalia Núñez
605      Esteban Quito
```

i després fes el mateix amb un registre que no existeix i mostra el diccionari resultant ...
Quins errors/excepcions has obtingut?

Fes els Exercicis

JAVA COLLECTIONS FRAMEWORK

Autor:
Àngel Olmos Giner, a partir de consultes en:
Documentació Oracle – Collections Framework Overview
Wikipedia - Java collections framework
Delftstack - La diferencia entre ArrayList y LinkedList en Java
Java T point - HashSet vs TreeSet