ArrayList: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/ArrayList.html
HashSet: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/HashSet.html
HashMap: https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/HashMap.html

ArrayList

1. Notes DAM

Volem calcular la nota mitjana de programació de la classe de 1r de DAM. Per fer-ho crearem un mètode introdueixValors on l'usuari introduirà notes a un ArrayList i acaba quan s'introdueix un -1.

A continuació hi haurà una funció sumaValors que sumarà tots els valors d'aquest ArrayList. (Fes el bucle amb Iterator i sense ell). Sense ell seria utilitzant el foreach, per exemple: for (Integer i: conjuntIntegers)

Finalment s'entrega la mitjana aritmètica a l'usuari.

2. Vaixells

Crea un petit programa per un port pesquer capaç de tenir una estructura ArrayList de Vaixells. Un Vaixell s'identifica per un nom, un preu i una edat.

Fes les operacions necessàries per introduir, buscar, modificar i treure un vaixell d'aquest ArrayList. També volem una opció on es llistin tots els vaixells actualment, a més volem que es llistin per antiguitat (de més vell a més nou), per ordre lexicogràfic de nom i per preu (de car a barat).

3. Sistema de gestió de jugadors

1. Classe de jugadors (Player): creeu una classe de jugador senzilla amb els atributs següents: name (cadena), score (int) i els mètodes necessaris.
2. Classe PlayerManager: creeu una classe PlayerManager amb les funcionalitats següents:
    1. afegeix un jugador a la llista.
    2. elimina un jugador de la llista en funció del nom del jugador.
    3. retornar el jugador en funció del nom del jugador.
    4. actualitza la puntuació d'un jugador.
    5. retornar una llista dels millors N jugadors en funció de les seves puntuacions.
    6. retornar una llista amb els jugadors amb score més gran que un valor minScore.

8. obtenir el jugador a més puntuació.
9. mostrar tots els jugadors actuals.

3. Creeu una classe TestPlayerManager per provar les funcionalitats anteriors, crea 5 jugadors.

4. Sistema de gestió de llibres.

1. Implementeu un classe anomenada Book amb els atributs següents: id (int), title (String), author (String), rating (int, valor entre 0 i 5 que representa la valoració d'un llibre), i els mètodes necessaris.

2. Classe BookManager: creeu una classe BookManager amb les funcionalitats següents:

1. afegir un llibre a la llista.
2. mostrar tots els llibres actuals.
3. retornar un llibre en funció del títol del llibre.
4. retornar un llibre en funció del autor del llibre.
5. actualitzar el rating d'un llibre.
6. retornar una llista ordenada descendent dels llibres en funció dels ratings.
7. retornar una llista dels millors N llibres en funció dels ratings.
8. retornar una llista amb els llibres amb rating més gran que un valor minRating.
9. eliminar un llibre de la llista en funció del id del llibre.

3. Creeu una classe TestBookManager per provar les funcionalitats anteriors, crea 5 llibres. S'ha de mostrar que totes les funcionalitats funcionen correctament. Afegiu println's perquè es vegi clarament el que s'està fent.

HashMap

1. Mòbils

Crea un Hashmap capaç d'inventariar tots els mòbils dels alumnes de DAM1. El Hashmap serà una estructura conjuntMobils<dni, Mobil>. Dni serà un String i classe Mobil està compost per una Marca, un model i un número.

Crea les següents opcions:

a) Afegir mòbil. (vigilar amb dni's repetits).

b) Treure mòbil.

c) Buscar un mòbil a partir d'un dni.

d) Buscar un dni a partir del número de mòbil.

2. Assignació de dorsals d'un equip de futbol.

Cal que definim una estructura de Hashmap <Dorsal, Jugador>, Un dorsal és només un enter, mentre que un Jugador és de la classe Jugador. Un Jugador conté un nom, una edat i una posició.

Feu un programa capaç de:

a) Introduir Jugadors al Hashmap.

b) Buscar un jugador per dorsal en el Hashmap.

c) Buscar un jugador per nom en el Hashmap.

d) Llistar Jugadors ordenats per número.

e) Llistar jugadors per edat.

f) Llistar jugadors per posició.

3. Sistema de matrícula del curs

1. Classe Student: creeu una classe d'estudiant amb els atributs següents:
    1. IdStudent: Identificador únic de l'estudiant.
    2. Name: Nom de l'alumne.
    3. Age: Edat de l'alumne.
2. Classe CourseManager: creeu una classe CourseManager amb les funcionalitats següents:
    1. enrollStudent(Student student): inscriu un estudiant al curs.
    2. unenrollStudent(String id): anul·la la matrícula d'un estudiant del curs en funció del seu
    identificador.
    3. findStudentById(String id): cerca i retorna un objecte d'estudiant pel seu identificador.
    4. listAllStudents(): enumera tots els estudiants inscrits al curs.

4. SCRUM

Volem gestionar els grups de SCRUM de DAM1 en un HashMap amb els noms dels equips i els alumnes membres de cada equip.

2.1 Classe Student. Implementeu la classe Student amb atributs id (String) i nom (String) amb els mètodes necessaris.
2.2 Classe Scrum. Implementeu la classe Scrum amb un Map<String,

ArrayList<Student>> (o HashMap) com atribut, els mètodes necessaris i les següents funcionalitats:

1. public boolean addProject(String projectName): aquest mètode ens permetrà afegir nous projectes. Si el projecte existeix, s'ha d'informar que ja existeix.
2. public boolean addStudent(String projectName, Student student): afegir un nou alumne a un projecte. Si el projecte no existeix, es crea un nou projecte i s'afegeix l'alumne.
3. public boolean removeProject(String projectName): esborrar un projecte. Si el projecte no existeix, s'ha d'informar que no existeix.
4. public void showProjects(): mostrar per pantalla els projectes actuals i els membres de cada projecte.
5. public boolean removeStudent(String projectName, String studentId): esborrar un alumne del projecte. S'ha de comprovar que el projecte existeix, i que el alumne forma part del projecte. Informar dels possibles casos.

Creeu una classe TestScrum per provar les funcionalitats anteriors. S'ha de mostrar que totes les funcionalitats funcionen correctament. Afegiu println's perquè es vegi clarament el que s'està fent.

HashSet

1. Loteria

Crea un sac amb boles d'una loteria. Aquest sac (Hashset) contindrà les boles del 0 al 100. Paral·lelament has de crear una classe Jugador que estarà compost d'un dni i un "boleto" amb 4 números del 0 al 100 diferents.

Crea 4 jugadors amb 4 boletos random i simula una loteria que vagi traient nombres del sac creat anteriorment. Un cop un dels jugadors guanya (s'han tret els 4 nombres del jugador) s'acaba el joc i es treu el dni del guanyador.

2. SCRUM

Fes un circuit amb diferents estructures de Hashset capaç de simular les fases per les que passa una tasca en un cicle d'SCRUM. Una tasca s'identifica per un nom i una descripció.

Cal tenir un conjunt de tasques de cada un d'aquests grups: TO BE DONE, DOING i DONE. Així doncs quan introduïm una tasca al sistema passa automàticament a TO BE DONE. Un cop s'està realitzant cal passar aquesta tasca de TO BE DONE a DOING, i un cop finalitzada passarà de DOING A DONE.

És necessari tenir aquestes opcions en el programa:

1. Introduir una tasca.
2. Passar de TO BE DONE A DOING
3. Passar de DOING A DONE
4. Mostrar l'estat actual dels tres conjunts de tasques.

3 Sentence Similarity.

En aquest exercici implementarem una funció que ens permetrà calcular la similitud entre dues frases (o dos documents de text), és a dir, com de properes o semblants són dues frases semànticament. Aquesta és una tasca fonamental dintre del camp de NLP (Natural Language Processing) que és una branca de la intel·ligència artificial, amb múltiples aplicacions. Aquí implementarem un mètode clàssic i senzill que avui dia gaire bé no es fa servir.

Escriu una classe anomenada **SentenceSimilarity** amb les següents funcions estàtiques:

3.1 Preprocessing. Escriu una funció anomenada **string2Set** que prengui una frase com a entrada i retorni un **Set<String>** amb les paraules d'aquesta frase. Es considera que les paraules de la frase estan separades per espais o signes de puntuació i no es té en comte majúscules i minúscules. Podeu simplificar i esborrar els signes de puntuacions.

Recordatori: podeu fer servir els mètodes split() i toLowerCase() de String per separar una frase en un array de paraules i convertir-la a lower case.

Signatura:

public static Set<String> string2Set(String sentence)

3.2 Intersection. Escriu una funció que prengui dues frases com a entrada i trobi la intersecció de paraules entre elles. Feu servir **string2Set** per fer el preprocessament.

Signatura:

public static Set<String> intersection(String sentence1, String sentence2): aquesta funció retorna un **Set<String>** amb les paraules comuns a les totes dues frases.

Podeu fer servir el mètode **retainAll()** de Collections per calcular la intersecció.

Exemple:

Input:

```
String sentence3 = "This a test";
String sentence4 = "this is another test";
```

Ouput:

[test, this]

3.3 Union. Escriu una funció que prengui dues frases com a entrada i trobi la unió de paraules entre elles, és a dir, totes les paraules que contenen les dues frases, sense repeticions. Feu servir **string2Set** per fer el preprocessament.

Signatura:

> **public static Set<String> union(String sentence1, String sentence2)**: aquesta funció retorna un **Set<String>** amb totes les paraules que contenen les dues frases, sense repeticions.

Exemple:

Input:

```
String sentence3 = "This a test";
String sentence4 = "this is another test";
```

Ouput:

[a, test, another, this, is]

3.4 Sentece Similarity. Ara implementarem una funció bàsica que calculi la similitud entre dues frases, que és una forma de mesurar com de semblants són dues frases (o documents de text en general). Farem servir un mètode clàssic de NLP (Natural Language Processing), tot i que avui dia es fan servir mètodes més eficients basats en Deep Learning.

Específicament ens demanen que implementem la funció de similitud de **Jaccard** (també coneguda com *intersection over union*) que es defineix com:

$$J(s1, s2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|} = \text{size of intersection(s1, s2) / size of union(s1, s2)}$$

és a dir, el nombre de paraules que ens dona intersection(s1, s2) dividit pel nombre de paraules que ens dona union(s1, s2). Aquesta funció ens donarà un nombre entre 0 i 1, de manera que com més similars siguin dues frases més a prop de 1 ens donarà la funció jaccard.

Signatura:

> **public static float jaccard(String sentence1, String sentence2)**

Exemple:

String s1 = "This a test";
String s2 = "this test is another test";

Intersection: [test, this] → size = 2
Union: [a, test, another, this, is] → size = 5

jaccard(s1, s2) = 0.4
jaccard(s1, s1) = 1
jaccard(s1, "") = 0

Al main() de SentenceSimilarity proveu amb algunes frases per comprovar que funciona correctament.

4. Seguiment del Jugadors

Crea un programa per gestionar el seguiment dels jugadors d'un joc. Hi han dos tipus de jugadors, lladres i policies.

Requeriments:

1. **Classe Player**:
    - `name` (nom del jugador).
    - `role` (pot ser "Thief" o "Police").
    - override de toString()
2. **Classe Game**:
    - Dos `HashSets`:
        - Un per a **active players** (jugadors actius que no estan eliminats).
        - Un per a **eliminated players** (jugadors eliminats).
    - Mètodes per:
        - Afegir un jugador al joc.
        - Eliminar un jugador pel seu nom (moure'l d'actius a eliminats).
        - Comprovar si el joc ha acabat (o bé si tots els Civilians o totes les Màfies estan eliminats).
        - Mostrar l'estat actual dels jugadors actius i eliminats.
3. **Classe GameTest**:
    - Simula les següents accions:
        - Afegir jugadors 5 jugadors de cada rol.
        - Eliminar jugadors.
        - Comprovar l'estat del joc.
        - Mostrar els jugadors actius i eliminats.