GOING INT'O SPACE WITH JAVA

minet

LES TYPES

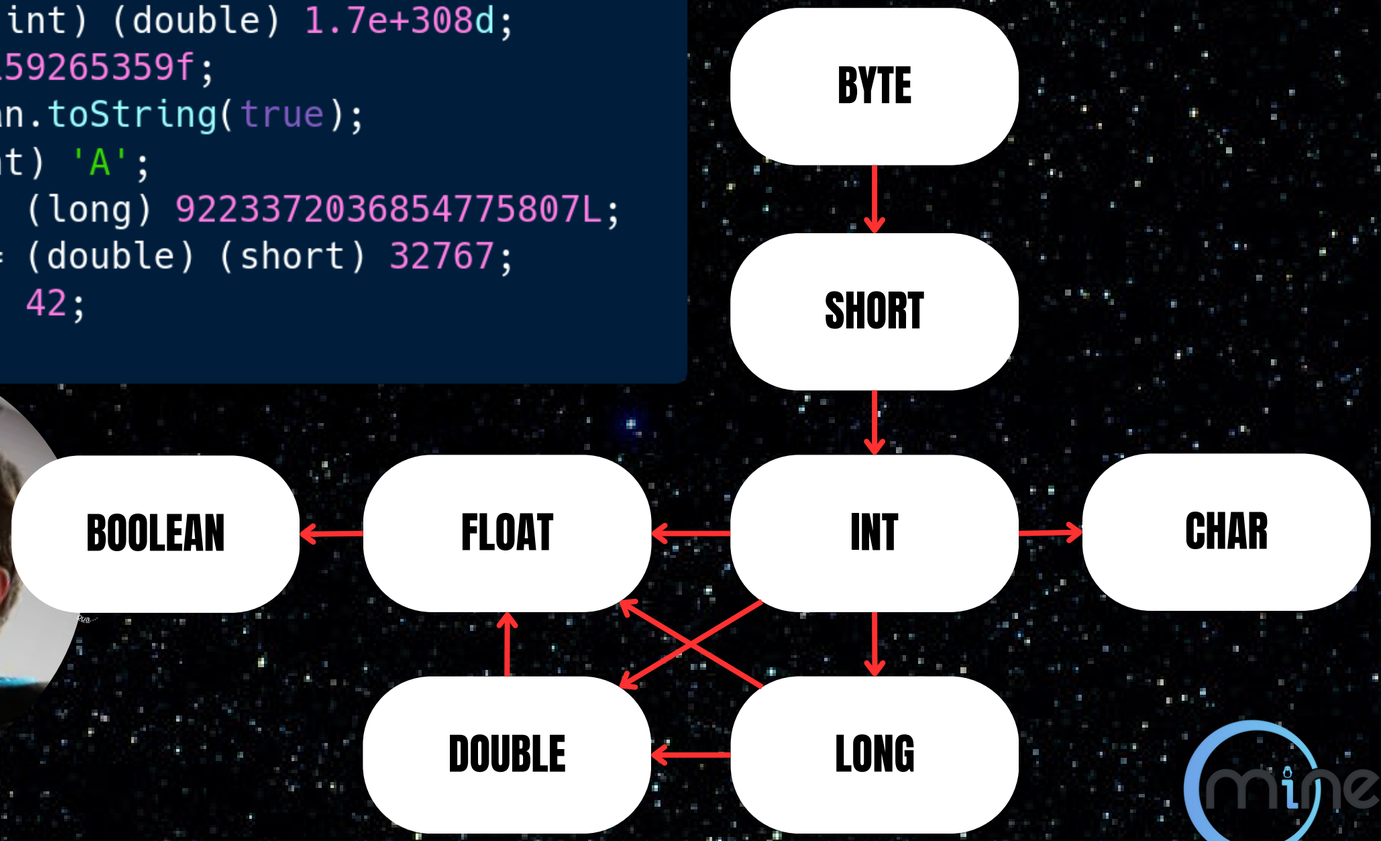
Type Primitifs

```
int codeCbNicolas = 1684;
double nbClubRayane = 1.7e+308d;
float pi = 3.14159265359f;
boolean validerCF = true;
char monGradeECTS = 'A';
long moneyINeed = 9223372036854775808L;
short SleepLossInHours = 32767;
byte everything = 42;
```

LES TYPES

conversion

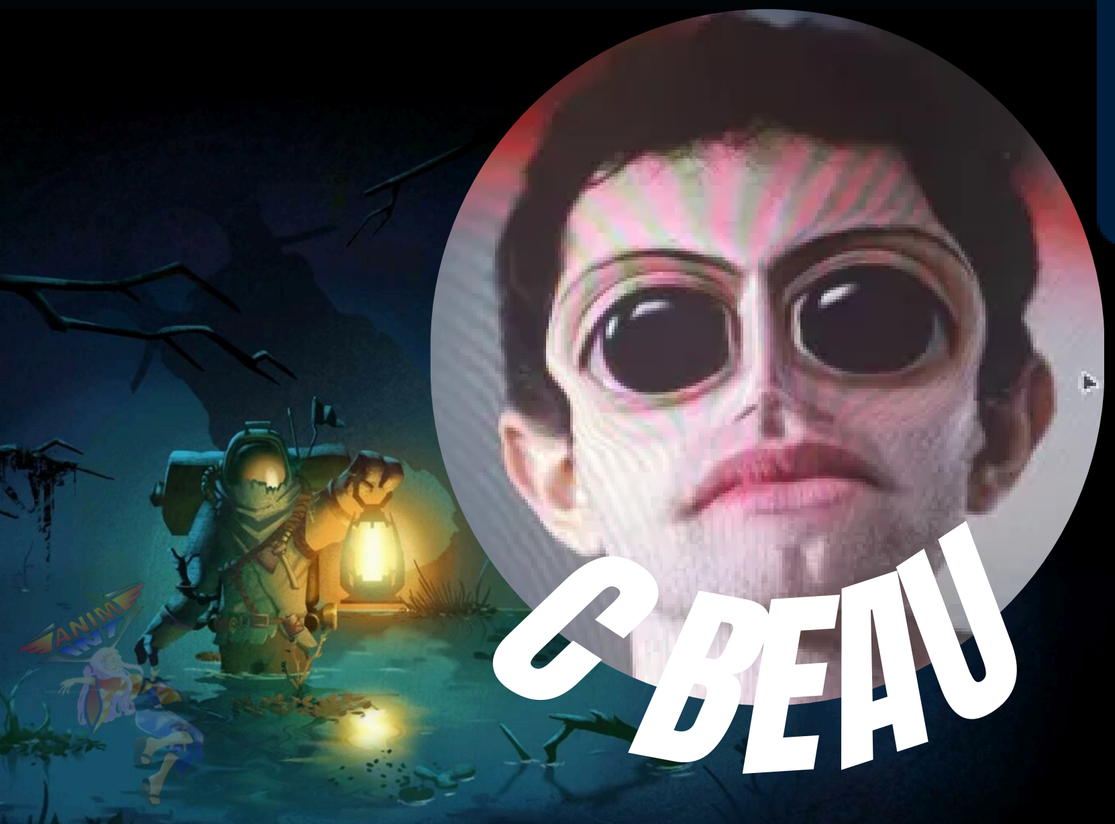
```
Double codeCbNicolasToDouble = (double) 1684;
Integer nbClubRayaneToInteger = (int) (double) 1.7e+308d;
Integer piToInteger = (int) 3.14159265359f;
String validerCFToString = Boolean.toString(true);
Integer monGradeECTSToAscii = (int) 'A';
Float moneyINeedToFloat = (float) (long) 92233720368547758O7L;
Double sleepLossInHoursToDouble = (double) (short) 32767;
Short everythingToShort = (short) 42;
```



TABLEAUX

```
int[] notes = {18,12,17}
int[] numbers = new int[5];
```

Type/Class

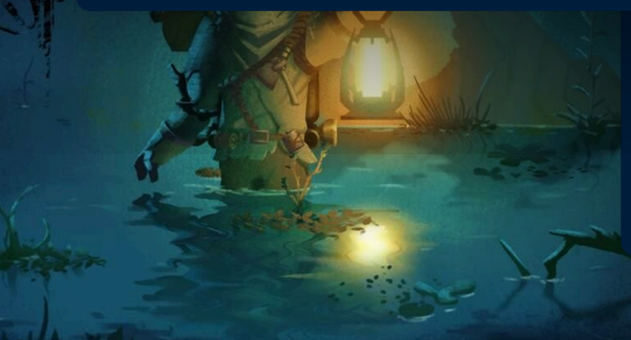


STRUCTURES CONDITIONNELLES

```java
int qi = 20;
if (qi >= 80) {
    System.out.println("You Dumb");
} else {
    System.out.println("NOT TOO STUPID");
}
```

```java
int passif = 0;
while (lister) {
  passif++;
}
```


```java
switch (action) {
    case "venir en forma":
        System.out.println("Valider");
        break;
    case "ne pas venir en forma":
        System.out.println("Pas Valider");
        break;
    case "venir dormir en forma":
        System.out.println("PAS Valider");
        break;
    default:
        System.out.println("skibidi");
}
```



```java
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

```java
do {
    System.out.println("be productive...");
    attempts++;
} while (attempts < 3);
System.out.println("Gave up.");
```

LES CLASS

```java
class MyCrush {
  private String name;

  public MyCrush(String name) {
    this.name = name;
  }
}
```


```java
public class ControlFinal {

    public ControlFinal() {
    }

    public static String staticModule(String param1, String param2)
        return "Static module called with param1=" + param1 + ", pa
    }

    private String instanceModule(String param1) {
        return "Instance module called with param1=" + param1;
    }

}
```

```java
Nicolas moi = new Nicolas()
```

```java
public static void main(String[] args)
```

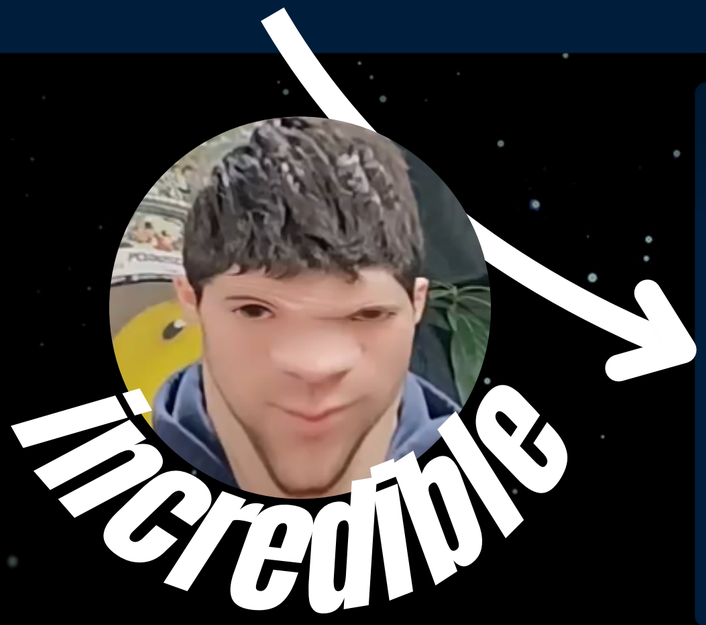
CONSTRUCTEUR

ACCESSIBILITÉ

PUBLIC, PRIVATE, PROTECTED

HÉRITAGE

```
class TaGrandMere extends Grandmother
```




```
class Grandmother {
    public void shareWisdom() {
        System.out.println("Grandmother's
            wisdom: Respect your roots.");
    }
    public void giveMoney() {
        System.out.println("Give lot's of
            money");
    }
}
```

INTERFACES

```java
interface Minetien {
    String beSmart();
    String haveProjetDev(Projet projet);
}
```

```java
public class Nicolas implements Minetien {

    @Override
    public String beSmart() {
        return "For this guy this not work";
    }

    @Override
    public String haveProjetDev(Projet projet) {
        return "Working on the project: " + projet.getName();
    }
}
```

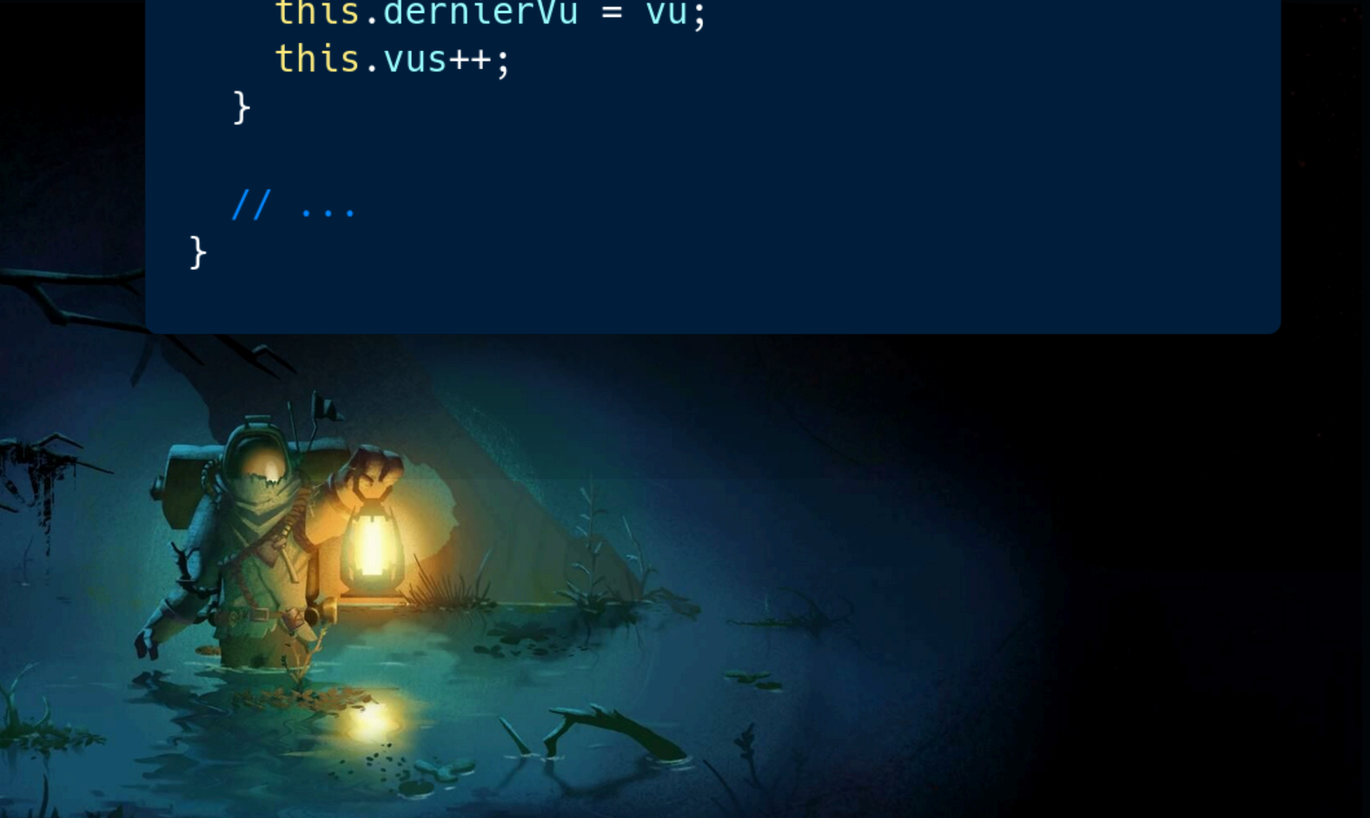
CLASSES GÉNÉRIQUES

```
class CompteurDeMoutons {
  private Mouton dernierVu;
  private int vus;

  public void voir(Mouton vu) {
    this.dernierVu = vu;
    this.vus++;
  }

  // ...
}
```

- 😢 TRISTOUNET
- 🐑 UTILISABLE UNIQUEMENT AVEC DES MOUTONS
- ⏳ NE PERMET PAS DE RÉUTILISER FACILEMENT

CLASSES GÉNÉRIQUES

```
class Compteur<T> {
  private T dernierVu;
  private int vus;

  public void voir(T vu) {
    this.dernierVu = vu;
    this.vus++;
  }

  // ...
}
```



- 😄 SYMPA ET AVENANT
- 🐙 UTILISABLE AUSSI AVEC DES PIEUVRES ET DES SOUS-MARINS
- 📲 RÉUTILISABLE DIRECTEMENT SANS MODIFICATION

CLASSES GÉNÉRIQUES

```java
class LoupGarou {/* Le village s'endort */}
class SousMarin {/* Bises sous-marines */}
class Compteur<T> {/* reste du code */}
// On peut aussi avoir plusieurs types :
class Autre<K, V extends Minetien>{/*  */}

// On spécifie T entre les <chevrons>
Compteur<LoupGarou> lycan =
  new Compteur<LoupGarou>();
// On peut enlever ↑ le type sera inféré
Compteur<SousMarin> hydra = new Compteur<>();

// Les tableaux ne prennent pas de chevrons
Autre<Int, Nicolas>[] tab = new Autre[10];
// ? Qu'est-ce qu'il y a dans ce tableau ?
tab[1] = new Autre<>(); // ça marche
```

EXCEPTIONS

```java
// Ouvre un fichier 1.txt
BufferedReader reader = new BufferedReader(new
FileReader("1.txt"));
// Écris le résultat dans un fichier 2.txt
BufferedWriter writer = new BufferedWriter(new
FileWriter("2.txt"));

String line;
while ((line = reader.readLine()) != null) {
    // Converti chaque ligne en entier naturel
    int number = Integer.parseInt(line);
    // Écris le résultat dans le fichier 2.txt
    writer.write(String.valueOf(number));
    writer.newLine();
}

// Ferme les fichiers
reader.close();
writer.close();
```

EXCEPTIONS

```java
// Vérifie si le fichier 1.txt existe
File inputFile = new File("1.txt");
if (!inputFile.exists()) {
  System.out.println("1.txt n'exist pas.");
  return;
}

// Ouvre le fichier 1.txt
BufferedReader reader = new BufferedReader(
  new FileReader(inputFile));
BufferedWriter writer = new BufferedWriter(
  new FileWriter("2.txt"));

String line;
// Lit le contenu ligne par ligne
while ((line = reader.readLine()) != null) {
  // Vérifie si la ligne est un entier valide
  if (line.matches("-?\\d+")) {
    int number = Integer.parseInt(line);
    // Vérifie si l'entier est naturel
    if (number >= 0) {
      // Écris le résultat dans le fichier
      writer.write(String.valueOf(number));
      writer.newLine();
    } else {
      System.out.println("Ligne invalide");
    }
  } else {
    System.out.println("Ligne invalide");
  }

}
```

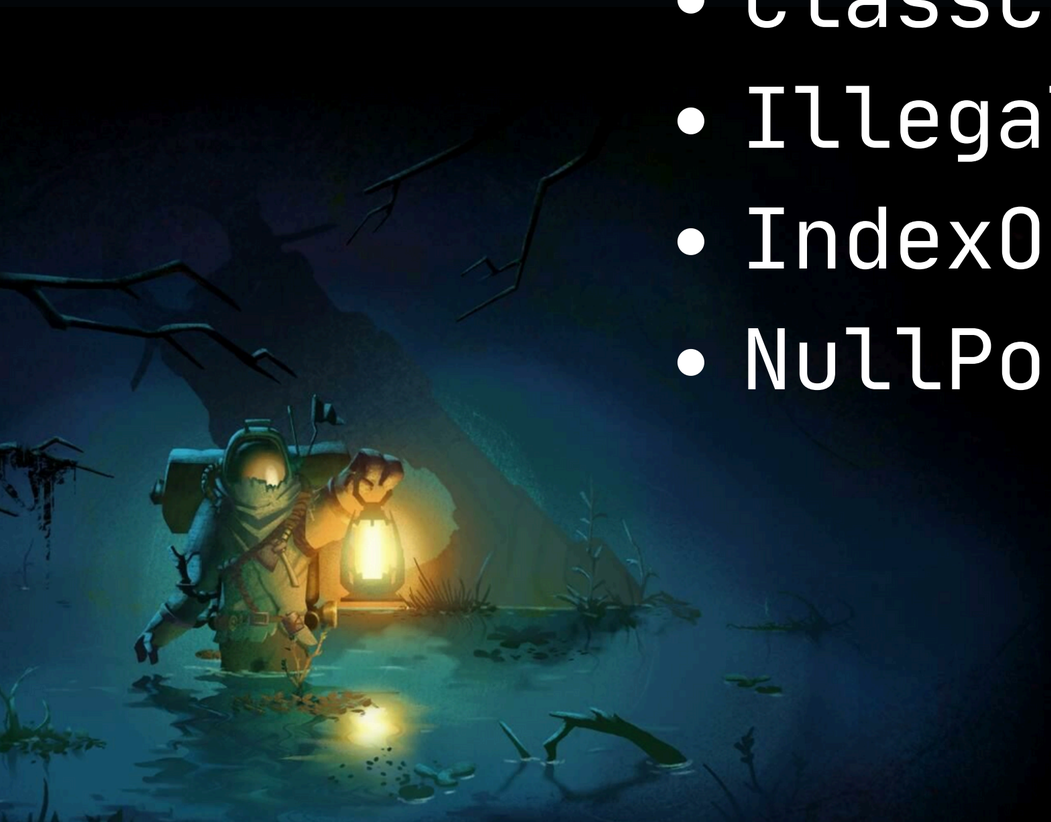
EXCEPTIONS


```java
// On définit une exception personnalisée
class NombreNegatifException extends Exception {}

try {
  BufferedReader reader = new BufferedReader(
    new FileReader("1.txt"));
  BufferedWriter writer = new BufferedWriter(
    new FileWriter("2.txt"));

  String line;
  while ((line = reader.readLine()) != null) {
    int number = Integer.parseInt(line);
    if (number < 0) {
      throw new NombreNegatifException();
    }
    writer.write(String.valueOf(number));
    writer.newLine();
  }

  reader.close();
  writer.close();

// Tous les cas particuliers sont à la fin
} catch (NumberFormatException e) {
  System.out.println(
    "Ligne invalide : pas un entier");
} catch (NombreNegatifException e) {
  System.err.println("Nombre négatif détecté");
} catch (IOException e) {
  // même ceux auxquels on a pas pensé
  System.err.println(e.getMessage());
}
```

EXCEPTIONS

À CONNAITRE :

- ClassCastException
- IllegalArgumentException
- IndexOutOfBoudsException
- NullPointerException



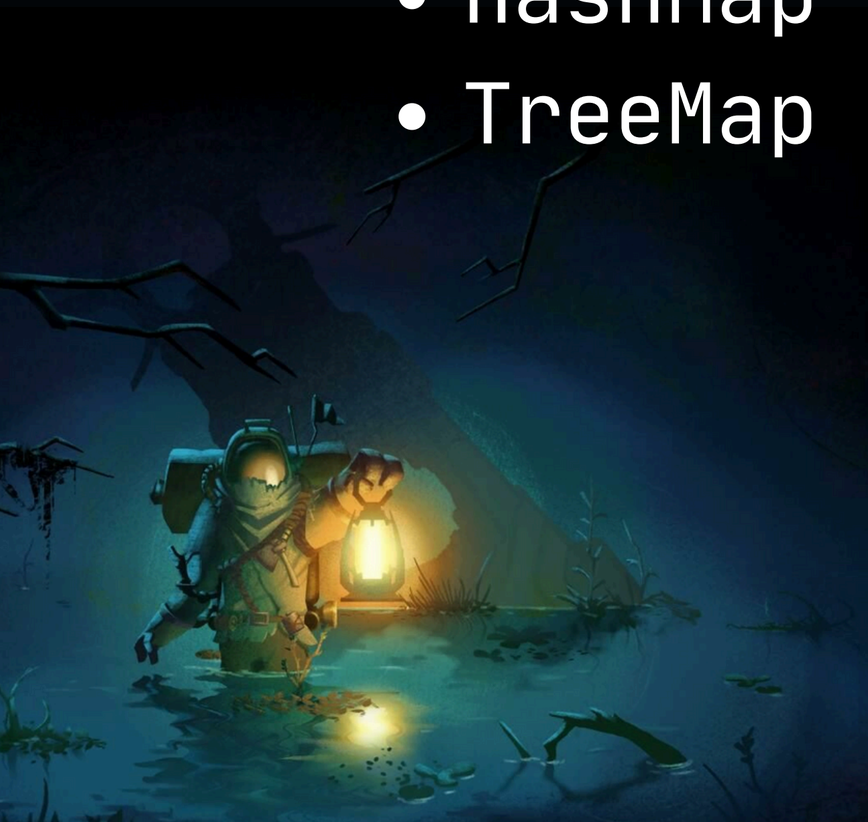
STRUCTURES DE DONNÉES

MAP :

- HashMap
- TreeMap

LISTES :

- ArrayList
- LinkedList



SET :

- HashSet
- TreeSet

QUEUE :

- LinkedList (encore ?)
- DequeArray

LES QUESTIONS
&
LE TP

LE TP

1. Écrire une classe Node<T> avec un champ data (T), frequency (int), real (boolean), left/right (Node<T>), un constructeur et des getters.
2. Écrire une classe NodeComp<T> avec l’interface Comparator avec une méthode compare qui renvoie la différence entre les fréquences de 2 Node passés en argument.
3. Dans le main, compter le nombre d’occurrences de chaque lettre dans la chaîne de caractères passée en argument.
4. Créer une PriorityQueue avec la classe NodeComp et ajouter un Node par lettre.
5. Tant qu’il reste plus d’un élément dans la queue, combiner les deux premiers Node dans un nouveau Node (real = false) avec un poids de left + right.

LE TP

6. Associer chaque lettre à son code (on obtient le code en parcourant l’arbre en se rappelant du chemin [left : 0, right : 1])
7. Coder la phrase passée en argument.
8. Vérifier le résultat en décodant la phrase en utilisant l’arbre, renvoyer une erreur NodeNotFoundException si la lettre ne fait pas partie de l’alphabet.

LA FIN VRAIE FIN

BON CF