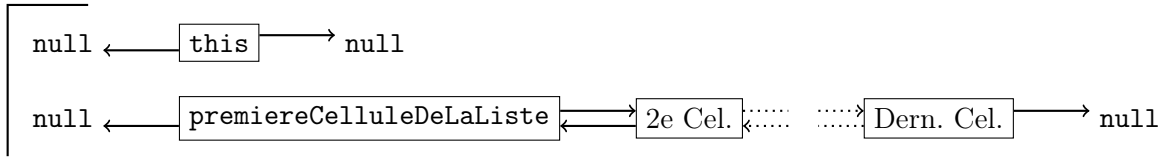
Le but de ce TP est d'implémenter une version simplifiée d'automate Cellulaire, vue comme une liste doublement *chaînée de booléens, qui indique si la cellule est vivante ou morte.*

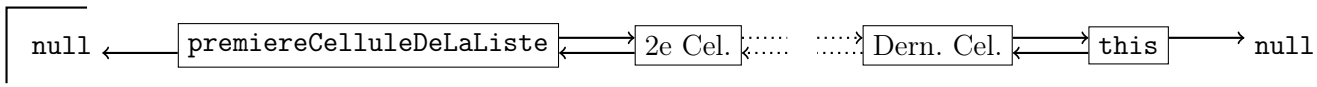
Exercice 1 [Liste doublement chaînée]

1. Créer une classe `Cellule` contenant :
   (a) Trois attributs : `precedente` & `suivante` de type `Cellule` ;
   `vivante` de type `boolean`.
   (b) Un getter pour l'attribut `vivante`.
   (c) Un getter et un setter pour les attributs `precedente` & `suivante`.
   (d) Un constructeur `Cellule(boolean vivante)` qui initialise l'attribut `vivante` avec l'argument, et les deux autres attributs à `null`.
   (e) Une méthode `void affiche()` qui imprime, *sans retourner à la ligne*, un dièse `#` si `vivante=true` et un tiret `-` si `vivante=false`.
2. Ajouter un constructeur `Cellule(boolean vivante, Cellule premiereCelluleDeLaListe)` qui ajoute la cellule en création (this) *à la fin* de la liste dont la première cellule est `premiereCelluleDeLaListe`.

avant



après

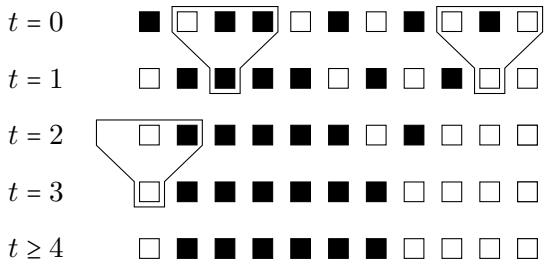


3. Tester tout ceci dans une classe principale `AutomateMain`, par exemple avec le code suivant :

```
Cellule liste = new Cellule (1); // crée une liste contenant une seule Cellule
new Cellule (0,liste);           // ajoute une Cellule à la fin de la liste
new Cellule (1,liste);           // ajoute une Cellule à la fin de la liste
liste.affiche();                                 // affiche #
liste.getSuivante().affiche();                   // affiche -
liste.getSuivante().getSuivante().affiche(); // affiche #
```

Il doit s'afficher : `#-#` .

La suite décrit le fonctionnement d'un automate cellulaire.
A chaque instant t, chaque cellule est soit vivante (noire dans les figures, `vivante=true` dans les programmes) soit morte (blanche dans les figures, `vivante=false` dans les programmes). L'état d'une cellule donnée à l'instant $(t+1)$ dépend de l'état de ses voisines et à l'instant t, ainsi que de son propre état à l'instant t.
Nous considérons d'abord la règle de la majorité, c'est-à-dire que la cellule d'indice i à l'étape $(t+1)$ est noire si au moins deux cellules parmi $(i-1)$, i et $(i+1)$ sont noires à l'étape t, voir figure ci-dessous. (Par convention, la cellule à gauche de la première (resp. à droite de la dernière) est toujours considérée comme morte).



Sont encadrés des exemples d'application de la règle de la majorité. À $t = 1$, la 3ème cellule est noire car, à $t = 0$, parmi les cellules 2, 3 et 4 la majorité sont noires. De façon analogue, l'avant-dernière cellule devient blanche à $t = 1$ car ses deux voisines sont blanches à $t = 0$.
Le troisième et dernier cadre souligne le fait que la première cellule considère que sa voisine de gauche est morte.

Exercice 2 [Automate]

1. Faire une classe `Automate` contenant :
   (a) Un attribut `Cellule premiereCellule`.
   (b) Un constructeur `Automate()` qui initialise à la liste vide.
   (c) Une methode `void initialisation()` qui initialise la liste comme à la figure précédente, à $t = 0$.
   (d) Une méthode `void affiche()` qui affiche les cellules en utilisant la méthode du même nom de la classe `Cellule`, puis retourne à la ligne.
2. Créer dans `AutomateMain` un `Automate`, l'initialiser (grâce à la méthode de la question 1c), puis l'afficher. Il doit s'afficher : `#-##-#-#-#-` .

Exercice 3 [Mise à jour]

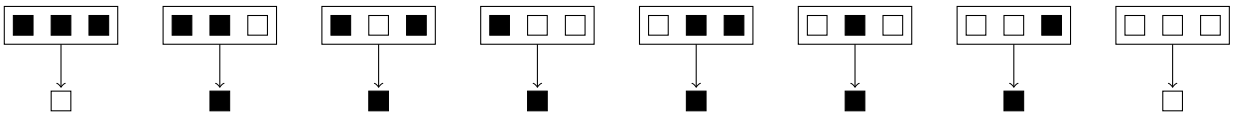
1. On veut créer une fonction qui change le statut `vivante` des cellules en fonction du temps. Il n'est pas possible de faire ceci avec un seul parcours de la liste : la mise à jour prématurée d'une cellule peut changer le résultat de de la mise à jour de sa voisine !
2. (a) Ajouter à la classe `Cellule` un attribut `prochainEtat` de type `boolean`. Changer les constructeurs pour que celui-ci soit toujours initialisé à `false`.
   (b) Ajouter dans la classe `Cellule` une méthode `prochaineEtape()` qui met `prochainEtat=true` si la cellule sera vivante à l'instant suivant (et `prochainEtat=false` si elle sera morte) en suivant la règle de la majorité.
   (c) Ajouter ensuite une méthode `void miseAJour()` qui met à jour la valeur de `vivante` à celle stockée dans `prochainEtat`.
   (d) Créer dans la classe `Automate` la méthode `uneEtape()` qui parcourt la liste *deux fois*, la première fois en appellant `prochaineEtape()` sur chaque cellule, la seconde fois en appelant `miseAJour()` sur chaque cellule.

(e) Tester : vous devez obtenir le résultat de $t = 1$ de la figure précédente.

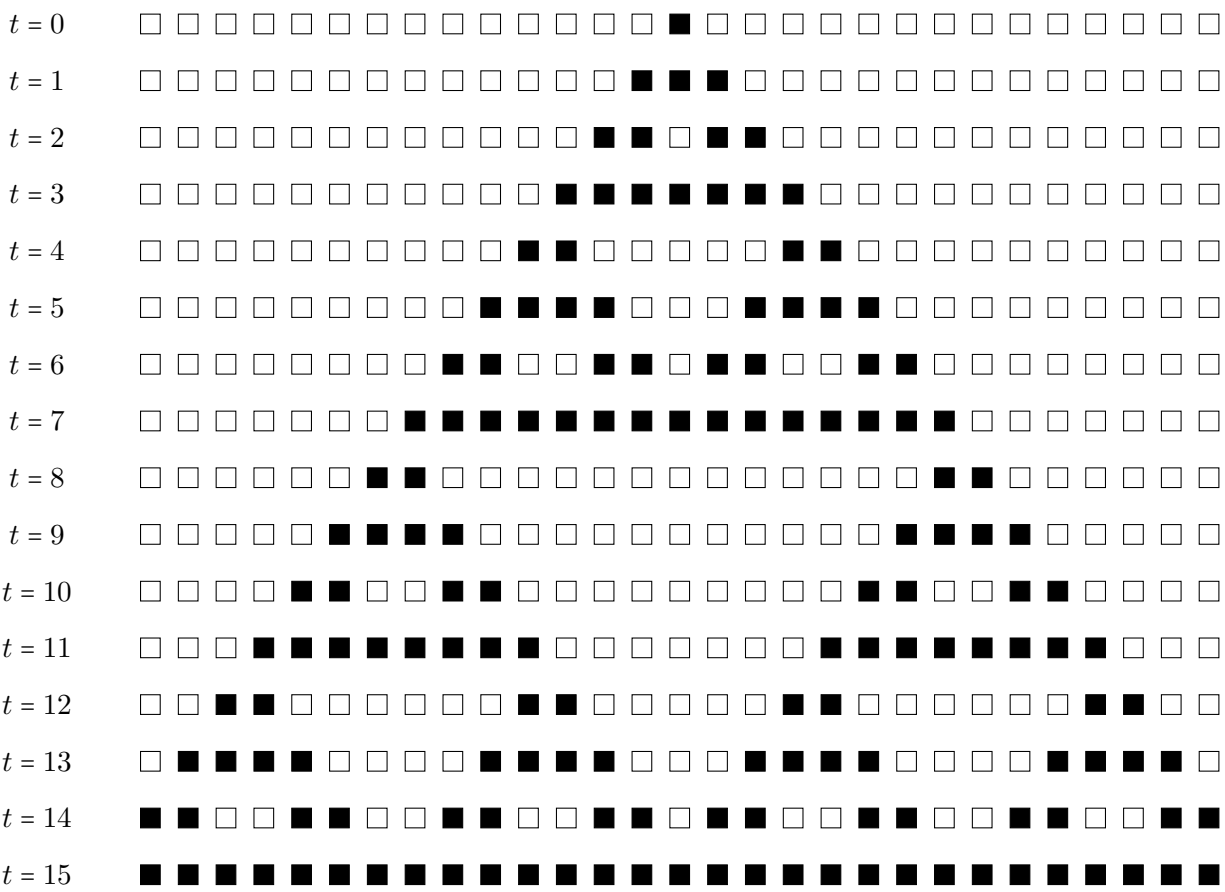
Exercice 4 [Quelques améliorations]

1. Ajouter à la classe `Automate`
   (a) Une méthode `void nEtapes(int n)` qui effectue n étapes successives, en affichant les étapes intermédiaires.
   (b) Tester avec $n = 5$ et comparer avec la figure de la page précédente.
   (c) Un constructeur `Automate(String str)` qui crée l'automate où, initialement, la cellule i est vivante si `str.charAt(i)=='#'`, ou est morte si `str.charAt(i)=='-'`.
   Par exemple, si l'on donne en argument la chaîne `#-##-#-#-#-` , on doit obtenir exactement le même état initial que précédemment.
   (d) Tester avec diverses chaînes de caractères.

Dans la suite, on veut utiliser des automates cellulaires qui suivent des règles plus complexes. Par exemple, la «règle 126» représentée à la figure suivante : une cellule devient blanche si, à l'étape précédente elle et ses voisines sont de la même couleur ; dans tous les autres cas, la cellule devient noire.



La figure suivante présente l'itération de cette règle sur un tableau qui ne comporte initialement qu'une seule cellule vivante. Il s'agit du triangle de Sierpinkski.



Exercice 5 [Règles plus complexes]

1. Ajouter à la classe `Automate`
   (a) un attribut `boolean[][][] regles` (tableau tridimensionnel d'entiers).
   Le sens à donner à ce tableau est que l'état de la cellule i à l'étape $(t+1)$ est égal à `regles[A][B][C]` où A, B et C sont les états respectifs, à l'étape t, des cellules $(i-1)$, i et $(i+1)$. Par exemple, pour la règle 126, `regles[0][0][0] = regles[1][1][1] = false` et toutes les autres cases du tableau tridimensionnel sont égaux à *true*.
   (N'oubliez pas de modifier tous les constructeurs d'`Automate` pour qu'ils initialisent cet attribut à `null`.)
   (b) un constructeur `Automate(String str, boolean[][][] regles)` dont la chaîne de caractère aura la même utilisation que pour l'Exercice 4, Question 1c.
2. Ajouter à la classe `Cellule`, une méthode `void prochaineEtape(int[][][] regle)` qui met dans `prochainEtat` l'état dans la cellule selon les règles données en arguments.
3. Modifier la méthode `uneEtape()` de la classe `Automate` pour qu'elle appelle la nouvelle méthode `prochaineEtape(this.regles)` de la question précédente au lieu de simplement `prochaineEtape()`.
4. Tester tout ceci en créant l'automate suivante la «règle 126» et dont l'état initial est une chaîne du type
   `----------#----------` .