efrei
PARIS PANTHÉON-ASSAS UNIVERSITÉ

Structure de données : Rapport sur les listes chaînées

Rédigé et présenté par :
- Cédric Alonso
- Jade Hatoum

Année académique 2023 - 2024

SOMMAIRE

I. Implémentation d’une Liste simplement chaînée

II. Gestion d’un Historique

III. Addition d’Entiers Très Larges

I. Implémentation d'une Liste simplement chaînée

1. Algorithme explicité

Structure Générale : Pensez à une liste chaînée comme une suite d'éléments connectés, où chaque élément, appelé *Cellule*, détient une valeur (une information qui peut être de n'importe quel type) et un lien vers la Cellule suivante dans la liste. La première *Cellule* de la liste est appelée la "tête", et la fin est la *Cellule* où le lien suivant pointe vers "NULL", indiquant la fin de la liste.

Nous avons décidé d'ajouter un lien supplémentaire permettant à une *Cellule* de connaître son précédent. Ainsi, nous sommes passés d'une liste chaînée simplement à une liste plus complexe doublement chaînée. Nous avons également choisi le langage Java afin d'implémenter notre liste chaînée en Objet.

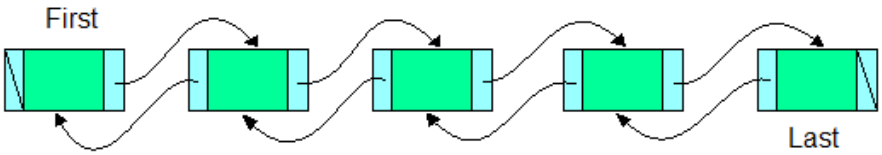


Schéma d'une liste chaînée bidirectionnelle

La méthode ajouter_debut est utilisée pour ajouter une nouvelle *Cellule* contenant une valeur spécifique au début de la liste chaînée. L'algorithme se déroule comme suit :

- Création d'une nouvelle *Cellule* avec la valeur donnée 'value'.
- Vérification de l'état de la liste. Si la liste est vide, la nouvelle *Cellule* devient à la fois la "tête" (début) et la "fin" de la liste.
- Si la liste n'est pas vide, les pointeurs de la nouvelle *Cellule* et de l'ancienne "tête" sont ajustés pour insérer la nouvelle *Cellule* au début de la liste.
- Augmentation de la taille de la liste de 1.

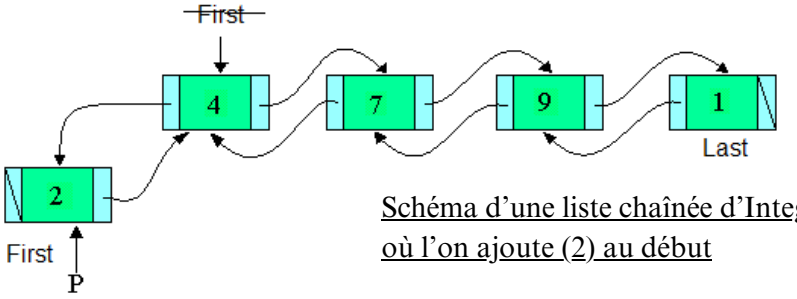


Schéma d'une liste chaînée d'Integer
où l'on ajoute (2) au début

La méthode ajouter_fin permet d'ajouter une nouvelle *Cellule* contenant une valeur spécifique à la fin de la liste chaînée. L'algorithme se déroule comme suit :

- Création d'une nouvelle *Cellule* avec la valeur donnée ‘value’.
- Vérification de l'état de la liste. Si la liste est vide, la nouvelle *Cellule* devient à la fois la "tête" (début) et la "fin" de la liste.
- Si la liste n'est pas vide, les pointeurs de la nouvelle *Cellule* et de l'ancienne "fin" sont ajustés pour insérer la nouvelle *Cellule* à la fin de la liste.
- Augmentation de la taille de la liste de 1.

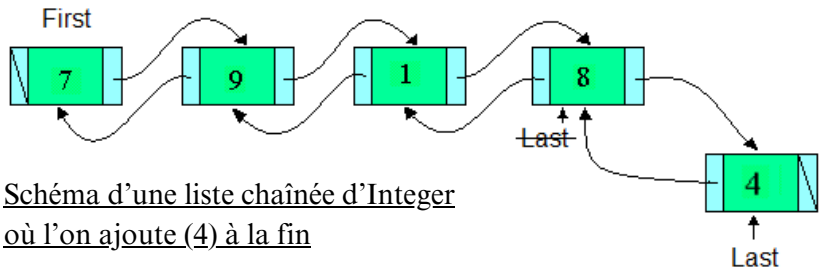


Schéma d’une liste chaînée d’Integer
où l’on ajoute (4) à la fin

La méthode rechercher permet de trouver une *Cellule* contenant une valeur spécifique dans la liste chaînée. L'algorithme se déroule comme suit :

- Initialisation de deux références, ‘currentStart’ au début de la liste et ‘currentEnd’ à la fin de la liste.
- Utilisation d'une boucle pour comparer la valeur recherchée value avec les valeurs des *Cellule* en se déplaçant simultanément depuis les deux extrémités de la liste vers le centre.
- À chaque itération, comparaison de la valeur value avec la valeur des *Cellule* pointées par ‘currentStart’ et ‘currentEnd’.
- Si une correspondance est trouvée avec l'une des *Cellule*, cette *Cellule* est renvoyée.
- Si la boucle atteint le point médian de la liste sans trouver de correspondance, on vérifie à nouveau la valeur de la *Cellule* pointée par ‘currentStart’, car la valeur recherchée peut se trouver au milieu de la liste.
- Si la valeur value est trouvée, la *Cellule* correspondante est renvoyée. Sinon, NULL est renvoyé pour indiquer que la valeur n'a pas été trouvée dans la liste.

Cette approche optimise la recherche en commençant des deux extrémités de la liste et en se déplaçant vers le centre, ce qui peut être plus rapide que de parcourir toute la liste dans le pire des cas.

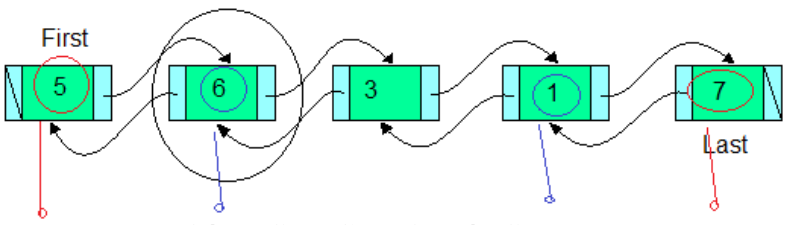


Schéma d’une liste chaînée d’Integer
où l’on recherche la valeur (6)

La méthode supprimer_debut est utilisée pour supprimer la première *Cellule* de la liste. L'algorithme se déroule comme suit :

- Vérification de l'état de la liste. Si la liste n'est pas vide, on procède à la suppression.
- Ajustement des pointeurs pour faire de la deuxième *Cellule* la nouvelle "tête" de la liste.
- Réinitialisation du pointeur précédent de la nouvelle "tête" pour indiquer la fin de la *Cellule* précédente.
- Réduction de la taille de la liste de 1.

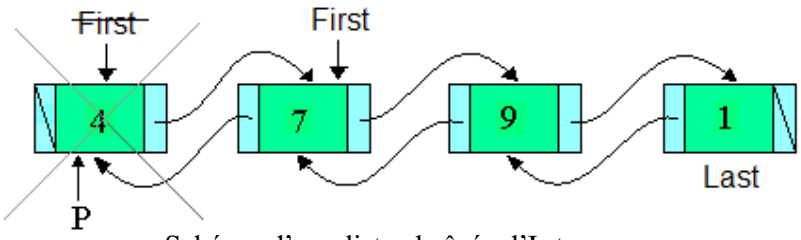


Schéma d'une liste chaînée d'Integer
où l'on supprime la tête

La méthode supprimer_fin est utilisée pour supprimer la dernière *Cellule* de la liste. L'algorithme se déroule comme suit :

- Vérification de l'état de la liste. Si la liste n'est pas vide, on procède à la suppression.
- Ajustement des pointeurs pour faire de la *Cellule* précédente la nouvelle "fin" de la liste.
- Réinitialisation du pointeur suivant de la nouvelle "fin" pour indiquer la fin de la *Cellule* actuelle.
- Réduction de la taille de la liste de 1.

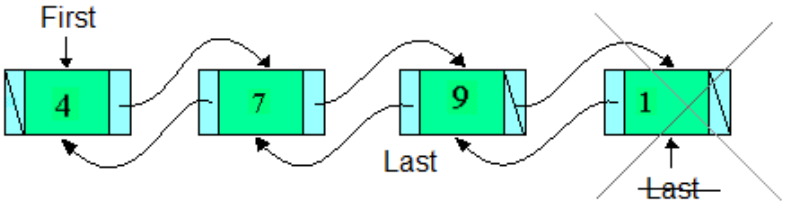


Schéma d'une liste chaînée d'Integer
où l'on supprime la fin

La méthode supprimer_val est utilisée pour supprimer une *Cellule* contenant une valeur spécifique de la liste chaînée. L'algorithme se déroule comme suit :

- Utilisation de la méthode rechercher pour localiser la *Cellule* à supprimer. Si la *Cellule* à supprimer n'est pas trouvée, aucune action n'est entreprise.

Si la *Cellule* à supprimer est trouvée, différentes actions sont entreprises en fonction de la position de la *Cellule* :

- Si la *Cellule* à supprimer est la "tête", les pointeurs sont ajustés pour faire pointer la "tête" vers la *Cellule* suivante en utilisant le même principe que supprimer_debut().
- Si la *Cellule* à supprimer est la "fin", les pointeurs sont ajustés pour faire pointer la "fin" vers la *Cellule* précédente en utilisant le même principe que supprimer_fin().
- Si la *Cellule* à supprimer se trouve entre deux autres *Cellules*, les pointeurs des Cellules voisines sont ajustés pour contourner la *Cellule* à supprimer.
- La *Cellule* à supprimer est dissociée de la liste en réinitialisant ses pointeurs.
- Réduction de la taille de la liste de 1.

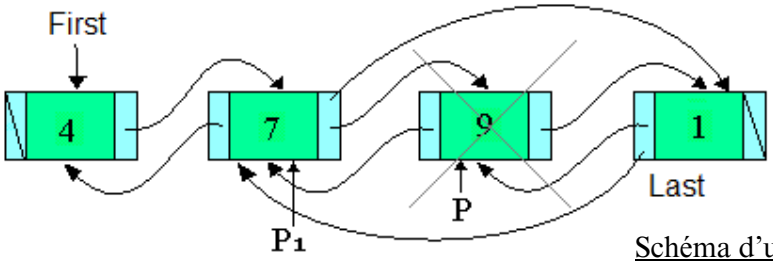


Schéma d'une liste chaînée d'Integer où l'on supprime la tête

2. Le code réalisé

```java
/**
 * Classe représentant une Cellule d'une liste chaînée.
 * @param <T> Le type de valeur stockée dans la cellule.
 * @version 1.0
 * @author Alonso Cédric
 * @author Hatoum Jade
 */
30 usages
public class Cellule<T>{
    /**
     * La valeur stockée dans la cellule.
     */
    5 usages
    private T value;

    /**
     * Le pointeur de la cellule suivante pour la cellule actuelle.
     */
    2 usages
    private Cellule<T> suivant;
```

```java
    /**
     * Le pointeur de la cellule précédente pour la cellule actuelle.
     */
    2 usages
    private Cellule<T> precedent;


    /**
     * Constructeur de la Cellule.
     * @param value La valeur à stocker dans la cellule.
     */
    4 usages
    public Cellule(T value){
        this.value = value;
    }

    7 usages
    public T getValue() {
        return value;
    }

    no usages
    public void setValue(T value) {
        this.value = value;
    }

    9 usages
    public Cellule<T> getSuivant() {
        return suivant;
    }

    7 usages
    public void setSuivant(Cellule<T> suivant) {
        this.suivant = suivant;
    }

    10 usages
    public Cellule<T> getPrecedent() {
        return precedent;
    }

    7 usages
    public void setPrecedent(Cellule<T> precedent) {
        this.precedent = precedent;
    }

    /**
     * Compare cette cellule avec un autre objet pour vérifier si les deux Cellules sont égales.
     * @param o L'objet à comparer.
     * @return true si les cellules sont égales, sinon false.
     */
    @Override
    public boolean equals(Object o){
        if (!this.getClass().equals(o.getClass()))
            return false;

        return this.value == ((Cellule<?>) o).getValue();
    }

    /**
     * Retourne une représentation en chaîne de caractères de la cellule.
     * @return La représentation de la cellule.
     */
    @Override
    public String toString(){
        return this.value.toString();
    }
}
```

```java
/**
 * Classe représentant une liste doublement chaînée générique.
 * @param <T> Le type de valeur stockée dans la liste.
 * @version 1.0
 * @author Alonso Cédric
 * @author Hatoum Jade
 */
10 usages
public class ListeChainees<T> {

    15 usages
    private Cellule<T> first;

    14 usages
    private Cellule<T> last;
    9 usages
    private int taille;

    /**
     * Constructeur par défaut de la liste chaînée initialisant une liste chaînée vide.
     */
    4 usages
    public ListeChainees(){
        this.first = null;
        this.last = null;
        this.taille = 0;
    }

    /**
     * Constructeur de la liste chaînée avec des paramètres initiaux.
     * @param first La première cellule de la liste.
     * @param last La dernière cellule de la liste.
     * @param taille La taille initiale de la liste.
     */
    no usages
    public ListeChainees(Cellule<T> first, Cellule<T> last, int taille){
        this.first = first;
        this.last = last;
        this.taille = taille;
    }

    /**
     * Ajoute une nouvelle cellule contenant une valeur spécifique au début de la liste.
     * @param value La valeur à ajouter.
     */
    4 usages
    public void ajouter_debut(T value){
        Cellule<T> toAdd;
        if (this.first != null) {
            toAdd = new Cellule<>(value);
            toAdd.setSuivant(this.first);
            this.first.setPrecedent(toAdd);
        }
        else {
            toAdd = new Cellule<>(value);
            toAdd.setSuivant(null);
            toAdd.setPrecedent(null);
            this.last = toAdd;
        }
        this.first = toAdd;
        this.taille++;

    }
```

```java
/**
 * Ajoute une nouvelle cellule contenant une valeur spécifique à la fin de la liste.
 * @param value La valeur à ajouter.
 */
5 usages
public void ajouter_fin(T value){
    Cellule<T> toAdd;
    if (this.last != null) {
        toAdd = new Cellule<>(value);
        toAdd.setPrecedent(this.last);
        this.last.setSuivant(toAdd);
    }
    else {
        toAdd = new Cellule<>(value);
        toAdd.setSuivant(null);
        toAdd.setPrecedent(null);
        this.first = toAdd;
    }
    this.last = toAdd;
    this.taille++;
}

/**
 * Recherche une cellule contenant une valeur spécifique dans la liste.
 * @param value La valeur à rechercher.
 * @return La cellule trouvée, ou null si la valeur n'est pas présente.
 */
3 usages
public Cellule<T> rechercher(T value){
    Cellule<T> currentStart = first;
    Cellule<T> currentEnd = last;
    int index = 0;

    while(taille/2 > index){
        if(currentStart.getValue() == value)
            return currentStart;

        if(currentEnd.getValue() == value)
            return currentEnd;

        currentStart = currentStart.getSuivant();
        currentEnd = currentEnd.getPrecedent();

        index++;
    }
    //if middle value
    if(currentStart.getValue() == value)
        return currentStart;

    return null;
}
```

```java
/**
 * Supprime la première cellule de la liste.
 */
2 usages
public void supprimer_debut(){
    if (first != null) {
        this.first = this.first.getSuivant();
        this.first.setPrecedent(null);
        taille--;
    }
}

/**
 * Supprime la dernière cellule de la liste.
 */
4 usages
public void supprimer_fin(){
    if (last != null) {
        this.last = this.last.getPrecedent();
        this.last.setSuivant(null);
        taille--;
    }
}

/**
 * Supprime une cellule contenant une valeur spécifique de la liste.
 * @param value La valeur à supprimer.
 */
1 usage
public void supprimer_val(T value) {
    Cellule<T> toDelete = rechercher(value);

    if (toDelete != null) {

        if (toDelete.equals(first))
            supprimer_debut();

        if (toDelete.equals(last))
            supprimer_fin();

        else {
            if (toDelete.getSuivant() != null) {
                toDelete.getSuivant().setPrecedent(toDelete.getPrecedent());
            }
            if (toDelete.getPrecedent() != null) {
                toDelete.getPrecedent().setSuivant(toDelete.getSuivant());
            }
        }

        toDelete.setPrecedent(null);
        toDelete.setSuivant(null);
        taille--;
    }
}
```

```java
    /**
     * Retourne une représentation en chaîne de caractères de la liste.
     * @return La représentation en chaîne de la liste.
     */
    public String toString(){
        StringBuilder sb = new StringBuilder();
        Cellule<T> current = this.first;
        do{
            sb.append(current);
            if (current.getSuivant() != null)
                sb.append("->");

            current = current.getSuivant();
        }while(current != null);

        return sb.toString();
    }

    2 usages
    public Cellule<T> getFirst() {
        return first;
    }

    4 usages
    public Cellule<T> getLast() {
        return last;
    }

    no usages
    public int getTaille() {
        return taille;
    }

}
```

3. Les tests et résultats de l'implémentation

```java
/**
 * Classe principale contenant la méthode main pour tester la classe {@link ListeChainees}.
 */
public class Main {
    public static void main(String[] args) {
        ListeChainees<Integer> liste = new ListeChainees<>();

        liste.ajouter_debut( value: 4);
        liste.ajouter_debut( value: 2);
        liste.ajouter_debut( value: 3);

        System.out.println("Liste apres ajout au debut : " + liste);

        liste.ajouter_fin( value: 5);
        liste.ajouter_fin( value: 1);

        System.out.println("Liste apres ajout a la fin : " + liste);

        int valeurRechercheeExist = 3;
        Cellule<Integer> maillonRecherche = liste.rechercher(valeurRechercheeExist);
        if (maillonRecherche != null) {
            System.out.println("Valeur " + valeurRechercheeExist + " trouvee dans la liste.");
        } else {
            System.out.println("Valeur " + valeurRechercheeExist + " non trouvee dans la liste.");
        }
```

```
        int valeurRechercheeNotExist = 7;
        maillonRecherche = liste.rechercher(valeurRechercheeNotExist);
        if (maillonRecherche != null) {
            System.out.println("Valeur " + valeurRechercheeNotExist + " trouvee dans la liste.");
        } else {
            System.out.println("Valeur " + valeurRechercheeNotExist + " non trouvee dans la liste.");
        }

        liste.supprimer_debut();
        System.out.println("Liste apres suppression au debut : " + liste);

        liste.supprimer_fin();
        System.out.println("Liste apres suppression a la fin : " + liste);

        int valeurSuppression = 2;
        liste.supprimer_val(valeurSuppression);
        System.out.println("Liste apres suppression de la valeur " + valeurSuppression + " : " + liste);
    }
}
```

```
Liste apres ajout au debut : 3->2->4
Liste apres ajout a la fin : 3->2->4->5->1
Valeur 3 trouvee dans la liste.
Valeur 7 non trouvee dans la liste.
Liste apres suppression au debut : 2->4->5->1
Liste apres suppression a la fin : 2->4->5
Liste apres suppression de la valeur 2 : 4->5

Process finished with exit code 0
```

II. Gestion d'un Historique

1. Algorithme explicité

Structure Générale : L'historique est une liste doublement chaînée, comme décrite précédemment à quelques détails près. Une cellule courante permet de connaître la position de l'utilisateur dans l'historique. Lors de l'ajout d'un URL à l'historique, ce dernier s'ajoute exclusivement à la fin de la liste chaînée. Uniquement le dernier URL de l'historique peut être supprimé. L'utilisateur peut se déplacer en avançant ou en reculant dans l'historique à raison d'un URL. Si l'utilisateur est au début de l'historique, alors il ne peut pas reculer, et s'il est à la fin, alors il ne pourra pas avancer. Ainsi comme notre classe ListeChainees permet de manipuler n'importe quel type de données, notre Historique traitera de 'String' représentant l'URL de la page web consultée.

La méthode ajouter_url est utilisée pour ajouter une nouvelle URL à la fin de l'historique de navigation.
L'algorithme se déroule comme suit :

- Il commence par vérifier si l'historique est vide en examinant la première cellule de la liste.
- Si l'historique est vide, une nouvelle cellule est créée avec l'URL donnée. Cette nouvelle cellule devient à la fois la "tête" (début) et la "fin" de l'historique, et la cellule courante est mise à jour pour pointer vers la nouvelle cellule.
- Si l'historique n'est pas vide, une nouvelle cellule est simplement ajoutée à la fin de la liste, sans affecter la cellule courante.
- Enfin, la taille de l'historique est augmentée de 1.

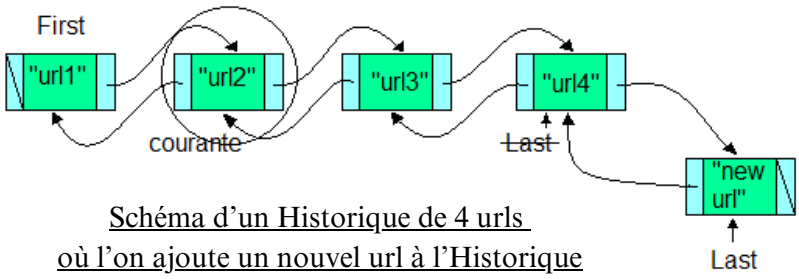


Schéma d'un Historique de 4 urls
où l'on ajoute un nouvel url à l'Historique

La méthode supprimer_dernier_url permet de supprimer la dernière URL de l'historique, en ajustant éventuellement la cellule courante.
L'algorithme se déroule comme suit :

- Si la cellule courante est la dernière cellule de l'historique, la cellule courante est mise à jour pour pointer vers la cellule précédente, puis la dernière cellule est supprimée de l'historique.
- Si la cellule courante n'est pas la dernière cellule, seule la dernière cellule est supprimée.
- Enfin, la taille de l'historique est réduite de 1.

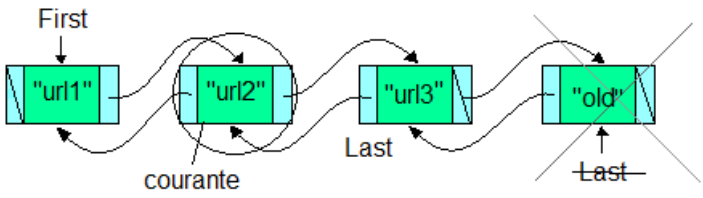


Schéma d'un Historique de 4 urls
où l'on supprime le dernier url (old)

La méthode avancer permet à l'utilisateur d'avancer d'une page dans l'historique, en mettant à jour la cellule courante.
L'algorithme se déroule comme suit :

- Il commence par vérifier si la cellule courante a un successeur (une cellule suivante). Si c'est le cas, la cellule courante est mise à jour pour pointer vers sa cellule successeur.
- Si la cellule courante n'a pas de successeur, une exception HistoriqueException est levée pour indiquer que l'utilisateur est déjà à la fin de l'historique.

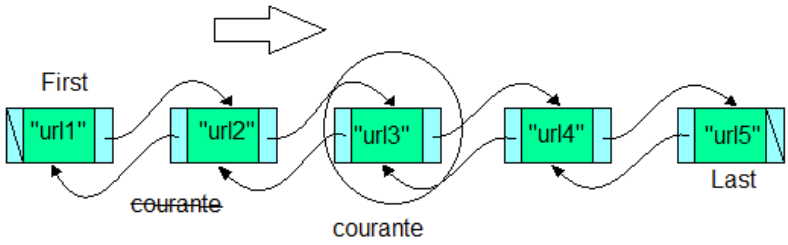


Schéma d'un Historique de 5 urls
où l'on avance à partir de l'url (2)

La méthode reculer permet à l'utilisateur de reculer d'une page dans l'historique, en mettant à jour la cellule courante.
L'algorithme se déroule comme suit :

- Il commence par vérifier si la cellule courante a un prédécesseur (une cellule précédente). Si c'est le cas, la cellule courante est mise à jour pour pointer vers sa cellule prédécesseur.
- Si la cellule courante n'a pas de prédécesseur, une exception HistoriqueException est levée pour indiquer que l'utilisateur est déjà au début de l'historique.

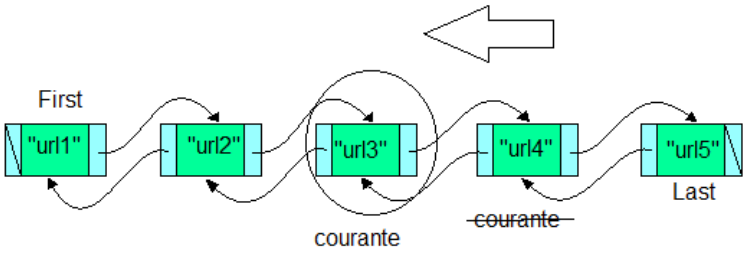


Schéma d'un Historique de 5 urls
où l'on recule à partir de l'url (4)

La méthode whereAmI renvoie une chaîne de caractères indiquant la position actuelle de l'utilisateur dans l'historique. Elle utilise la valeur stockée dans la cellule courante pour indiquer l'URL actuellement visité.

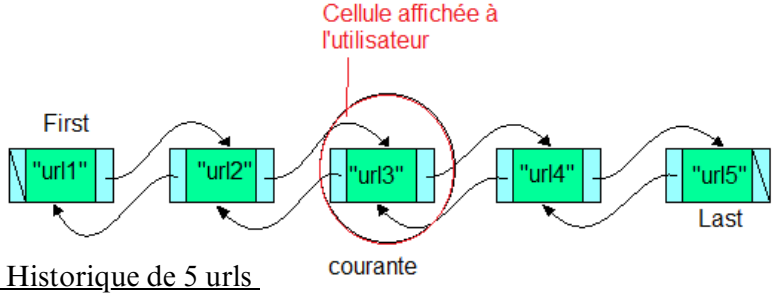


Schéma d'un Historique de 5 urls
où la cellule actuelle est l'url (3)

2. Le code réalisé

```java
/**
 * Classe représentant un historique de navigation web basé sur une liste chaînée {@link ListeChainees}.
 *
 * @version 1.0
 * @author Alonso Cédric
 * @author Hatoum Jade
 * */
2 usages
public class Historique {
    /**
     * La liste chaînée utilisée pour représenter l'historique.
     */
    10 usages
    private final ListeChainees<String> historique;

    /**
     * La cellule courante dans la liste, indiquant la position actuelle de l'utilisateur.
     */
    10 usages
    private Cellule<String> courant;

    /**
     * Constructeur par défaut de la classe Historique.
     * Initialise un nouvel historique vide.
     */
    1 usage
    public Historique(){
        this.historique = new ListeChainees<>();
    }

    /**
     * Ajoute une URL à l'historique, la plaçant à la fin de la liste.
     * Si l'historique est vide, la cellule courante est également mise à jour.
     * @param url L'URL à ajouter à l'historique.
     */
    4 usages
    public void ajouter_url(String url){
        if (this.historique.getFirst() == null){
            this.historique.ajouter_fin(url);
            this.courant = this.historique.getFirst();
        }
        else
            this.historique.ajouter_fin(url);
    }

    /**
     * Supprime la dernière URL de l'historique, ajustant la cellule courante si nécessaire.
     */
    1 usage
    public void supprimer_dernier_url(){
        if (this.courant.equals(historique.getLast())){
            this.courant = historique.getLast().getPrecedent();
            this.historique.supprimer_fin();
        }
        else
            this.historique.supprimer_fin();
    }

    /**
     * Avance d'une page dans l'historique, mettant à jour la cellule courante.
     * @throws HistoriqueException Si l'utilisateur est déjà à la fin de l'historique.
     */
    5 usages
    public void avancer() throws HistoriqueException{
        if (this.courant.getSuivant() != null)
            this.courant = this.courant.getSuivant();
        else
            throw new HistoriqueException("Vous ne pouvez pas avancer, Vous etes deja a la fin de l'historique");
    }
```

```java
    /**
     * Recule d'une page dans l'historique, mettant à jour la cellule courante.
     * @throws HistoriqueException Si l'utilisateur est déjà au début de l'historique.
     */
    2 usages
    public void reculer() throws HistoriqueException{
        if (this.courant.getPrecedent() != null)
            this.courant = this.courant.getPrecedent();
        else
            throw new HistoriqueException("Vous ne pouvez pas reculer, vous etes deja au debut de l'historique");
    }

    /**
     * Renvoie une chaîne de caractères indiquant la position actuelle de l'utilisateur.
     * @return Une chaîne de caractères représentant l'emplacement actuel de l'utilisateur.
     */
    8 usages
    public String whereAmI(){
        return "Vous etes actuellement ici : " + this.courant.getValue();
    }

    /**
     * Renvoie une représentation sous forme de chaîne de caractères de l'historique complet.
     * @return Une chaîne de caractères représentant l'historique complet.
     */
    public String toString(){
        return historique.toString();
    }
}

public class HistoriqueException extends Exception {

    2 usages
    public HistoriqueException(String message) {
        super("\u001B[31m" + message + "\u001B[0m");
    }

}
```

3. Les tests et résultats de l'implémentation

```java
public class Main {
    public static void main(String[] args) {
        Historique historique = new Historique();

        historique.ajouter_url("Google.com");
        historique.ajouter_url("Netflix&Chiiiiiiiiiilll.com");
        historique.ajouter_url("Wikipedia.org");
        historique.ajouter_url("MyEfrei.yes");

        System.out.println(historique);

        System.out.println(historique.whereAmI());

        try {
            historique.reculer();
        } catch (HistoriqueException e) {
            System.out.println(e.getMessage());
        }

        try {
            historique.avancer();

        } catch (HistoriqueException e) {
            System.out.println(e.getMessage());
        }

        System.out.println(historique.whereAmI());

        try {
            historique.avancer();
        } catch (HistoriqueException e) {
            System.out.println(e.getMessage());
        }
        System.out.println(historique.whereAmI());
```

```
        try {
            historique.reculer();
        } catch (HistoriqueException e) {
            System.out.println(e.getMessage());
        }
        System.out.println(historique.whereAmI());

        try {
            historique.avancer();
        } catch (HistoriqueException e) {
            System.out.println(e.getMessage());
        }
        System.out.println(historique.whereAmI());

        try {
            historique.avancer();
        } catch (HistoriqueException e) {
            System.out.println(e.getMessage());
        }
        System.out.println(historique.whereAmI());

        try {
            historique.avancer();
        } catch (HistoriqueException e) {
            System.out.println(e.getMessage());
        }

        System.out.println(historique.whereAmI());

        historique.supprimer_dernier_url();

        System.out.println(historique);

        System.out.println(historique.whereAmI());
    }
}
```

```
Google.com->Netflix&Chiiiiiiiiiiilll.com->Wikipedia.org->MyEfrei.yes
Vous etes actuellement ici : Google.com
Vous ne pouvez pas reculer, vous etes deja au debut de l'historique
Vous etes actuellement ici : Netflix&Chiiiiiiiiiiilll.com
Vous etes actuellement ici : Wikipedia.org
Vous etes actuellement ici : Netflix&Chiiiiiiiiiiilll.com
Vous etes actuellement ici : Wikipedia.org
Vous etes actuellement ici : MyEfrei.yes
Vous ne pouvez pas avancer, Vous etes deja a la fin de l'historique
Vous etes actuellement ici : MyEfrei.yes
Google.com->Netflix&Chiiiiiiiiiiilll.com->Wikipedia.org
Vous etes actuellement ici : Wikipedia.org

Process finished with exit code 0
```

III. Addition d'Entiers Très Larges

1. Algorithme explicité

Structure Générale : Afin de traiter certains entiers très larges de type *long*, nous ferons recours à la classe Entier, dépendant de notre classe ListeChainees, et donc Cellule également. Ces dernières nous permetteront de reprendre leurs propriétés et la nature de leur structure dans le but de diviser puis traiter ces entiers en composant plus simples, voire élémentaires. La classe Entier ne gardera que deux attributs : une liste chainées de *long* appelée *entier*, ainsi que une valeur *int* appelée *nbChiffresParCellule*, qui comme son nom l'indique, précise le nombre de chiffres maximal enregistrée dans une des cellules de la liste chainée *entier.*

Lors de l'instanciation, deux constructeurs sont proposés. Le premier ne prend que le *nbChiffresParCellule* en paramètre, et attribue une liste chainées vide à *entier.* Quant au second, partage les mêmes propriétés que le premier mais permet cette fois-ci de préciser la valeur en *long,* puis procède au formatage de la liste chainées *entier* à travers la méthode init.

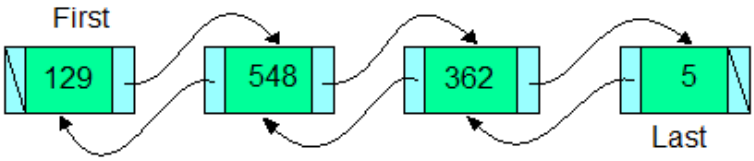


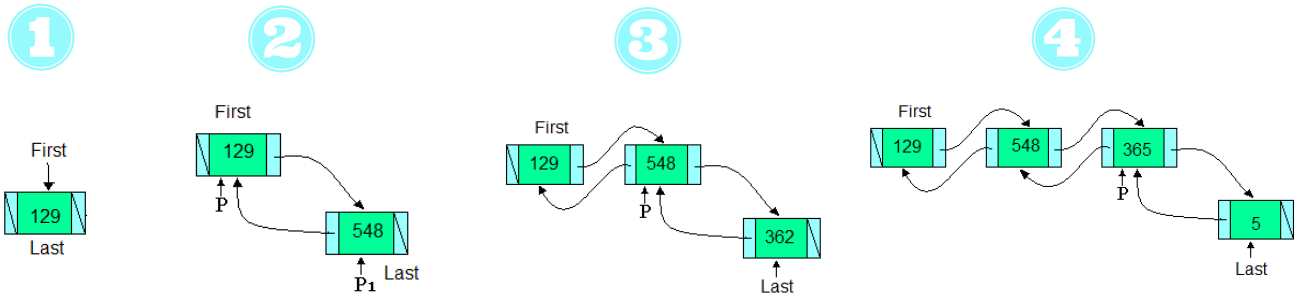
Schéma d'une instance de Entier:
nbChiffresParCellule = 3
valeur d'entrée : 1295483625L

La méthode init est conçu pour l'initialisation de l'attribut *entier* contenant donc la structure coeur de la classe Entier. Précisémment, l'algorithme suit les étapes suivantes pour achever ceci :

- Conversion de *value* en *String* dans *sEntier*, afin de pouvoir itérer chaque charactère plus tard, donc chaque chiffre dans notre cas.
- Création d'une liste de *String* temporaire *nbs,* qui réservera chaque substring de taille *nbChiffresParCellule.*
- Dans une boucle, attribution de tous les substrings de *sEntier,* à l'exception de la dernière, dans *nbs.*
- Si *sEntier* ne possède pas une taille multiple de *nbChiffresParCellule*, rajouter le dernier substring négligé lors de la boucle précédente dans *nbs.*
- Finalement, reconvertir chaque substring dans *nbs* en *long,* pour les stocker dans l'attribut *entier,* grâce à la méthode ajouter_fin.

Entier à stocker dans la liste chaînée avec 3 chiffres par cellules

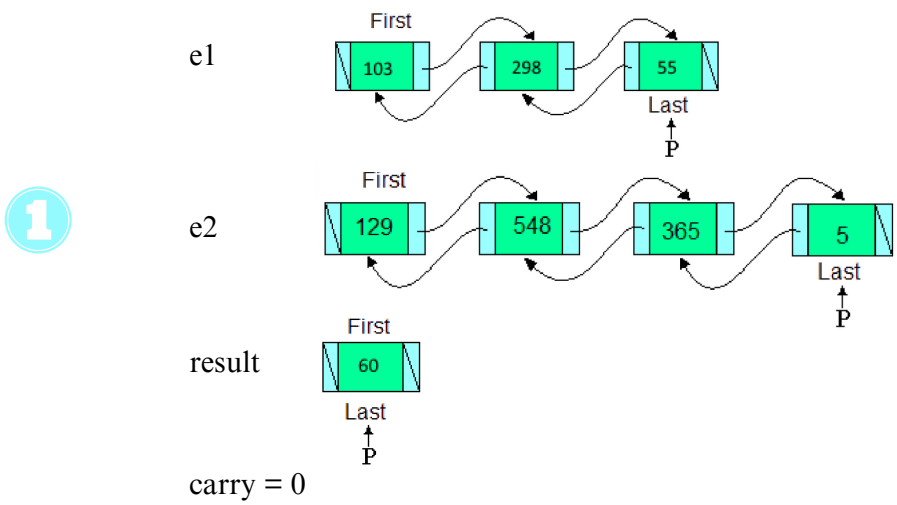
1295483625 ⟶ 129 | 548 | 362 | 5



Étapes d'initialisation de la découpe d'un entier dans une liste chaînée avec 3 chiffres par cellules

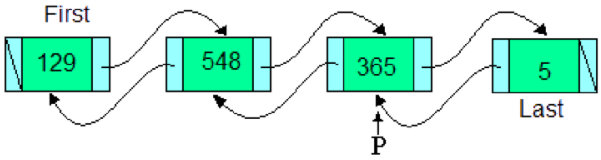
La méthode additionner est responsable de l'action principale de la classe Entier. En effet, elle permet d'additionner les *long* stockés dans les *Cellule* des listes chainées des instances Entier *this* (l'objet courant) et *entierToAdd* (l'objet en paramètre à additionner), et retourne donc cette somme *result* sous type Entier également. L'algorithme suit les étapes suivantes :

- Initiation de l'Entier *result*, en utilisant le constructeur ne prenant en paramétre que l'attribut *nbChiffresParCellule* et non pas la valeur *long,* puisque ce dernier n'a pas encore été calculé.
- Initiation des derniers *Cellule* de *this* et *entierToAdd,* ainsi que l'*int carry*, représentant la retenue. Cette dernière a pour but de conserver la valeur en overflow, limité par *nbChiffreParCellule.*
- Itération continue tant qu'aucunes des *Cellule* précédentes de *this* et *entierToAdd* ne sont nulles, ou qu'il n'y ait plus de retenue *carry* à traiter. Dans le cas où la retenue n'est rajoutée à aucune autre somme des cellules de *this* et *entierToAdd*, une *Cellule* sera consacrée à elle uniquement.
  - Rajout de la retenue de l'itération précédente à la somme *sum* de l'itération actuelle.
  - Si la valeur de la *Cellule* étudiée n'est pas nulle, additionner la valeur de celle-ci, avec *getValue*, avec la somme *sum* de l'itération actuelle.
  - Attribution de la retenue de la somme euclidienne à *carry* pour la prochaine itération.
  - Attribution de la somme finale sans overflow au *long cellValue.*
  - Finalement, rajout de la valeur au début de la liste chainée *entier* de *result.*

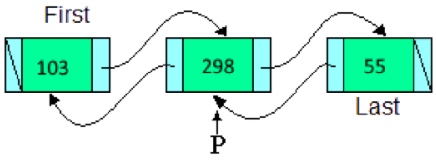


2

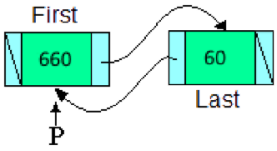
e1



e2



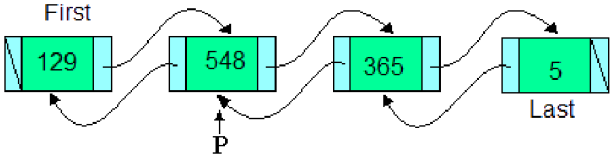
result



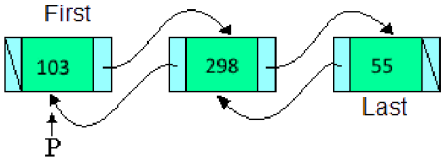
carry = 0

3

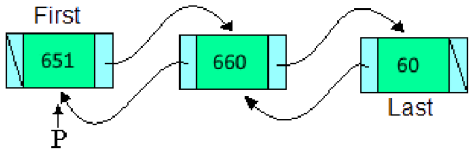
e1



e2



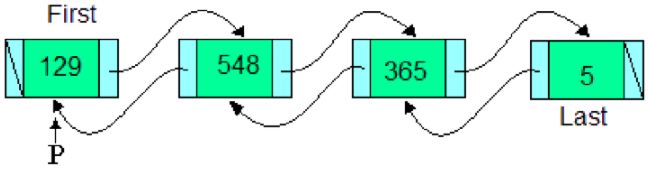
result



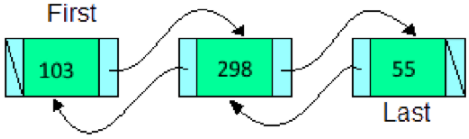
carry = 0

4

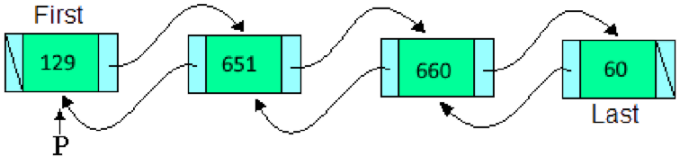
e1



e2



result



carry = 0

2. Le code réalisé

```java
/**
 * Classe représentant un nombre entier de grande taille basé sur une liste chaînée {@link ListeChainees}.
 *
 * @version 1.0
 * @author Alonso Cédric
 * @author Hatoum Jade
 */
10 usages
public class Entier {

    /**
     * La liste chaînée utilisée pour représenter l'entier.
     */
    6 usages
    private final ListeChainees<Long> entier;

    /**
     * Le nombre de chiffres par cellule dans l'entier.
     */
    9 usages
    private final int nbChiffresParCellule;

    /**
     * Constructeur de la classe Entier.
     *
     * @param nbChiffresParCellule Le nombre de chiffres par cellule.
     */
    1 usage
    public Entier(int nbChiffresParCellule){
        this.entier = new ListeChainees<>();
        this.nbChiffresParCellule = nbChiffresParCellule;
    }
    /**
     * Constructeur de la classe Entier initialisé avec une valeur initiale.
     *
     * @param value La valeur initiale de l'entier.
     * @param nbChiffresParCellule Le nombre de chiffres par cellule.
     */
    2 usages
    public Entier(Long value, int nbChiffresParCellule){
        this.entier = new ListeChainees<>();
        this.nbChiffresParCellule = nbChiffresParCellule;

        this.init(value);
    }

    /**
     * Initialise l'entier en découpant la valeur en chiffres d'une longueur donnée.
     *
     * @param value La valeur initiale de l'entier.
     */
    1 usage
    private void init(long value){
        String sEntier = String.valueOf(value);
        int i;
        List<String> nbs = new ArrayList<>();
        for (i = 0; i+nbChiffresParCellule-1 < sEntier.length(); i+=nbChiffresParCellule)
            nbs.add(sEntier.substring(i, i+nbChiffresParCellule));

        if (sEntier.length() % nbChiffresParCellule != 0)
            nbs.add(sEntier.substring(i));

        for (String nb : nbs)
            if (!nb.isEmpty())
                this.entier.ajouter_fin(Long.valueOf(nb));
    }
```

```java
/**
 * Additionne cet entier avec un autre entier donné.
 *
 * @param entierToAdd L'entier à ajouter.
 * @return Un nouvel {@link Entier} représentant la somme des deux entiers.
 */
1 usage
public Entier additionner(Entier entierToAdd) {
    Entier result = new Entier(nbChiffresParCellule);

    Cellule<Long> currentThis = this.entier.getLast();
    Cellule<Long> currentOther = entierToAdd.getEntier().getLast();

    int carry = 0;

    while (currentThis != null || currentOther != null || carry > 0) {
        long sum = carry;

        if (currentThis != null) {
            sum += currentThis.getValue();
            currentThis = currentThis.getPrecedent();
        }

        if (currentOther != null) {
            sum += currentOther.getValue();
            currentOther = currentOther.getPrecedent();
        }

        carry = (int) (sum / Math.pow(10, nbChiffresParCellule)); // Calculate the carry for the next iteration

        long cellValue = sum % (long) Math.pow(10, nbChiffresParCellule);

        result.getEntier().ajouter_debut(cellValue);
    }

    return result;
}
```

```java
    public ListeChainees<Long> getEntier() {
        return entier;
    }

    /**
     * Renvoie la liste chaînée représentant cet entier.
     *
     * @return La liste chaînée représentant l'entier.
     */
    public String toString(){
        return this.entier.toString();
    }
}
```

3. Les tests et résultats de l'implémentation

```java
public class Main {

    public static void main(String[] args) {
        Entier e1 = new Entier( value: 10329855L,  nbChiffresParCellule: 3);

        Entier e2 = new Entier( value: 1295483625L,  nbChiffresParCellule: 3);

        Entier e3 = e1.additionner(e2);

        System.out.println("e1 : " + e1);
        System.out.println("e2 : " + e2);

        System.out.println(e1 + " + " + e2 + " = " + e3);
    }
}
```

```
e1 : 103->298->55
e2 : 129->548->362->5
103->298->55 + 129->548->362->5 = 129->651->660->60

Process finished with exit code 0
```

efrei
PARIS PANTHÉON-ASSAS UNIVERSITÉ

Année académique 2023 - 2024