C. Sempé

# Liste Chaînée et récursivité

Nous voulons créer une classe Liste avec la technique objet de l'héritage, voici son implémentation.

```
public abstract class Liste
{
   public abstract boolean estVide();
   public abstract String getPremier();
   public abstract Liste getReste();
}

class ListeVide extends Liste
{
   ListeVide ()
   {
   }

   public boolean estVide(){  return true;}

   public String getPremier(){return null;}

   public Liste getReste(){return null;}
}

class ListeCons extends Liste
{
   private String valeur;
   private  .....   suiv;

   ListeCons(String val, Liste L)
   {
      this.valeur = val; this.suiv = L;
   }

   public boolean estVide(){ return false;  }

   public String getPremier() { return this.valeur; }

   public Liste getReste() { return this.suiv; }

}
```

Nous voulons chaîner les éléments de la liste.

- Observez la déclaration de la structure de données. Complétez la déclaration. Que va contenir cette liste chaînée ?
- Représentez graphiquement une liste avec trois éléments. Comment se terminera la liste ?

Le constructeur

Pour construire une liste nous pouvons écrire :

```
L = new ListeCons( "Un", new ListeCons( "Deux", new ListeCons( "Trois", new ListeCons( "Quatre", new ListeVide() ) )));
```

- Quel objet est instancié en premier ? Dans quel ordre est construite la liste ? Quel est le type de la variable L ?

Méthodes

A quoi servent les Méthodes getPremier() et getReste(). Que renvoient-elles ?

Nous allons mettre en place un certain nombre de fonctions supplémentaires utiles à la manipulation des listes.

## 1. Parcourir une liste en itératif

- Faites la méthode L.afficheIt(), cette méthode est abstract dans la classe Liste.
  Affichez la liste avec une boucle.

- Faites la méthode L.rechercherIt( val ) -> booléen

## 2. Parcourir une liste en récursif

Nous allons utiliser la récursivité pour développer ces fonctions.

Pour chaque fonction définir
- l'arrêt de la récursion.
- le traitement général faisant intervenir le ou les appels récursifs.

### *a. Méthode size()*

Nous allons développer la fonction L.size() en récursif

Si L={ A, B, C, D } alors L.size() = 4

Arrêt de la récursion

- Donnez la condition pour laquelle la fonction size() n'a pas besoin de faire d'appel récursif ?

Traitement général

- On remarque que { A, B, C, D}.size() = 1 + .....

- Écrivez la méthode size() et faites la trace des appels pour { A, B, C, D}.size()

### *b. Méthode somme()*

- Faites la méthode {A, B, C, D }.somme()→ "ABCD" et faire la trace.
  Rappel : vous pouvez cumuler dans une variable S de type String des éléments S = S+valeur

### c. *Afficher une liste*

- Faites la méthode afficher()
- Faites la méthode afficherInverse()

### d. *Méthode copie()*

- Faites la méthode {A, B, C, D }.copie() -> {A, B, C, D }
  (faire la trace des appels)

### e. *Méthode placerFin()*

- Faites la méthode {A, B, C, D }.placerFin( E) -> {A, B, C, D, E }

## TP

- Créez un nouveau projet ListeChainee. Placez ensuite le fichier Liste.java dans le dossier src de l'arborescence.
- Complétez cette classe avec les nouvelles méthodes ci-dessous.

```
public abstract boolean trouver( String val );

public abstract int  size();

public abstract Liste concat( Liste L1 );

public abstract Liste inverser();    (utilisation de placerFin() ! )

public abstract Liste inserer(String s , int rang );
```

- Proposez le main pour tester TOUTES ces méthodes.

Ajoutez les outils suivants (ils utilisent la méthode trouver() ):

```
public abstract Liste intersection ( Liste L)

public abstract Liste union ( Liste L) // sans doublon.

public abstract boolean inclus( Liste L)
```

# Dessins récursifs

Végétaux

- Ouvrez les fichiers vegetal.java, Fenetre.java, compilez et exécutez DemoFenetre.java.

Le programme dessine simplement deux ramifications.

- Observez et comprenez le calcul effectué, placez des commentaires.

- Modifiez la méthode Branche() pour qu'elle dessine récursivement cette figure:

(Les segments sont réduits de 15 pixels à chaque étapes) Pour maîtriser l'arrêt de la récursivité, vous devrez vous aider d'un paramètre.

Il est intéressant de ralentir le dessin afin observer l'ordre dans lequel le dessin s'effectue.

- Placez judicieusement plusieurs appels à la méthode Attendre() pour ralentir le dessin.
- Modifiez la méthode Branche() pour dessiner la figure de la **droite vers la gauche**.
- De la même manière modifiez la méthode pour que le dessin se construise à partir **des feuilles vers la racine**.

- Modifiez la méthode pour placer sur chaque feuille de l'arbre un fruit.

(Utilisez la méthode Fruit() )

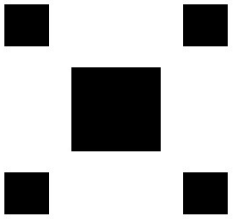
- Modifiez la méthode pour avoir des fruits sur chaque noeud de l'arbre.

- "Plantez" d'autres végétaux sur la fenêtre. Utilisation des coordonnées de la souris ? Voir mouseDown….

Des figures

- Utilisez le dossier Formes avec les fichiers qu'il contient.

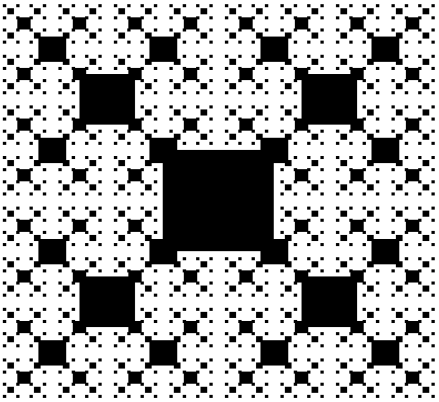
- Utilisez la classe Forme qui contient une méthode récursive Figure() qui réalisera le dessin à droite.



public void Figure( int x, int y, int largeur, int profondeur, Graphics g )

La figure de base vous est donnée ici après une étape récursive.

Le résultat après six récursions



Changez la couleur de chaque rectangle avec setColor() et AleaColor().

L'utilisateur choisit l'endroit du dessin :

- Déclarez deux variables d'instances X, Y qui seront initialisées à 300 à l'initialisation de la fenêtre.
- Ces variables seront mises à jour, par la méthode mouseDown() qui reçoit comme arguments le couple (x,y), représentant les coordonnées du pointeur de souris lors du clic.
- Paint() dessine la figure en X,Y
- Testez

Décor Disco:

Pour tapisser la fenêtre avec plusieurs Formes, on va donner la possibilité à l'utilisateur de les placer avec la souris.

Paint() doit donc redessiner les Formes à chaque fois. Il faut donc mémoriser les coordonnées (x,y) de ces formes pour que Paint() puisse les régénérer à chaque fois.

- Mettez en place dans la fenêtre cette nouvelle fonctionnalité.

- Limitez à 8 le nombre de figures possibles.

Inventez de nouvelles figures récursives ...