Questions de révision pour l'examen final.

Voir à la fin des exercices pour le code des classes ..

1. Donnez ce qui sera affiché après l'exécution des instructions suivantes

```
Maillon<Integer> p = new Maillon<Integer> ( 3 );
Maillon<Integer> w = new Maillon<Integer> ( 4, p );
p = new Maillon<Integer> ( 5, w );
w = new Maillon<Integer> ( 6, p.suivant() );
p.modifierSuivant ( w );
while ( p != null ) {
    System.out.println ( p.info() );
    p = p.suivant();
}
```

2. Dessinez les maillons produits par l'exécution des instructions suivantes :

```
Maillon<Integer> p = new Maillon<Integer> ( 1 );
Maillon<Integer> w = new Maillon<Integer> ( 2 );
Maillon<Integer> x = new Maillon<Integer> ( 3, w );
p.modifierSuivant ( x.suivant() );
x.suivant().modifierSuivant ( w );
p.modifierSuivant ( new Maillon<Integer> ( 4, w ) );
Maillon<Integer> a = new Maillon<Integer> ( 5, x.suivant().suivant() );
w.modifierSuivant ( p.suivant() );
```

3. Écrire une méthode qui prend en paramètre la référence vers le premier maillon d'une liste et qui retourne le nombre de maillons que contient la liste. Voici l'entête de la méthode :

```
public static int nbMaillons ( Maillon debut )
```

4. Écrire une méthode qui prend en paramètre la référence vers le premier maillon d'une liste et qui retourne la référence vers le premier maillon d'une nouvelle liste qui contient le même nombre de maillons avec les mêmes informations, mais en ordre inverse. Voici l'entête de la méthode :

```
public static Maillon copieInverse ( Maillon debut )
```

5. Faites la même chose qu'en 4 mais les maillons copiés doivent contenir les mêmes informations dans le même ordre. Voici l'entête de la méthode :

```
public static Maillon copieMemeOrdre ( Maillon debut )
```

6. Écrire une méthode qui prend en paramètre la référence vers le premier maillon d'une liste de nombres entiers et qui retourne la référence vers le premier maillon de cette même liste dans laquelle on aura retiré les maillons contenant un nombre négatif. Si la liste ne contient que des nombres négatifs, la liste sera donc vide et `null` devra être retourné. Voici l'entête de cette méthode :

```
public static Maillon<Integer> enleverNegatifs ( Maillon<Integer> debut )
```

7. Considérez les 24 éléments suivants (leur indice est inscrit au dessus en italique) :

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
1	2	5	7	12	17	18	19	20	25	26	28	35	40	42	43	53	56	57	61	64	73	77	87

(a) En utilisant la recherche séquentielle **non-ordonnée**, donnez les positions consultées lors de la recherche de 25.

(b) En utilisant la recherche séquentielle **non-ordonnée**, donnez les positions consultées lors de la recherche de 30.

(c) En utilisant la recherche séquentielle **ordonnée**, donnez les positions consultées lors de la recherche de 25.

(d) En utilisant la recherche séquentielle **ordonnée**, donnez les positions consultées lors de la recherche de 30.

(e) En utilisant la recherche **binaire**, donnez les positions consultées lors de la recherche de 19.

(f) En utilisant la recherche **binaire**, donnez les positions consultées lors de la recherche de 50.

8. Considérez le tableau suivant :

0	1	2	3	4	5	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
50	17	-5	39	63	55	103	99	60	12	6	57	77	8	81	50	26	82	7	90	-13	11	63

(a) Pour les tris **insertion**, **sélection** et **bulle**, faites la trace en donnant l'état du tableau après chaque tour de la boucle principale.

(b) Faites la trace du **Quicksort** en donnant l'état du tableau après la partition (à la fin de l'étape de la partition, l'élément pivot se retrouve à la droite de la portion de gauche).

9. Supposons que les données à trier sont déjà triées. Parmi les 4 algorithmes de tris vus au cours (insertion, sélection, bulle, quicksort), lequel sera le plus efficace? Justifiez.

10. Expliquez pourquoi la recherche séquentielle ordonnée peut être plus efficace que la recherche séquentielle non ordonnée dans certains cas.

11. Pourquoi la recherche binaire ne s'utilise-t-elle pas avec des listes chaînées?

12. Considérez les données suivantes

```
25 39 30 17 10 15 27 100 12 5 37 200 35 34 8 32 150 125 26 120 112 198 160 36 9 180 107 130 1 22 170 33
```

(a) Insérez-les dans un arbre binaire de recherche (dans l'ordre où elles apparaissent ci-haut). Dessinez cet arbre.

(b) Quelles sont les valeurs rencontrées lors de la recherche de 34.

(c) Quelles sont les valeurs rencontrées lors de la recherche de 155.

(d) Donnez les parcours préfixe, infixe et postfixe de l'arbre obtenu en (a).

(e) Définissez les termes suivants : feuille, nœud, hauteur, racine, sous-arbre.

(f) Considérez l'arbre obtenu en (a), donnez le nombre de feuilles, le nombre de nœuds, la hauteur de l'arbre.

(g) Quel est le nombre minimal de comparaisons effectuées lors de la recherche d'un élément absent de l'arbre.

(h) Quel est le nombre maximal de comparaisons effectuées lors de la recherche d'un élément absent de l'arbre.

13. Écrire une méthode qui prend en paramètre la référence vers le nœud racine d'un ABR et qui retourne le nombre de nœuds que contient l'ABR. Voici l'entête de la méthode :

```
public static int nbNoeuds ( Noeud racine )
```

14. Écrire une méthode qui prend en paramètre la référence vers le nœud racine d'un ABR et qui retourne la hauteur de celui-ci. S'il s'agit d'un arbre vide, la méthode doit retourner -1. Voici l'entête de la méthode :

```
public static int hauteur ( Noeud racine )
```

15. Écrire une méthode qui prend en paramètre la référence vers le nœud racine d'un ABR et qui retourne le nombre de feuilles que contient l'ABR. Voici l'entête de la méthode :

```
public static int nbFeuilles ( Noeud racine )
```

16. Écrire une méthode qui prend en paramètre la référence vers le nœud racine d'un ABR et qui retourne le nombre de nœuds à un enfant que contient l'ABR. Voici l'entête de la méthode :

```
public static int nbNoeuds1enfant ( Noeud racine )
```

17. Écrire une méthode qui prend en paramètre la référence vers le nœud racine d'un ABR et qui retourne le nombre de nœuds à deux enfants que contient l'ABR. Voici l'entête de la méthode :

```
public static int nbNoeuds2enfants ( Noeud racine )
```

18. Écrire une méthode qui prend en paramètre la référence vers le nœud racine d'un ABR ainsi qu'un entier. La méthode doit retourner le nombre de nœuds supérieurs ou égaux à l'entier passé en paramètre. Tentez de faire une méthode efficace (il n'est pas nécessaire dans tous les cas de visiter chaque nœud de l'arbre). Voici l'entête de la méthode :

```
public static int nbNoeudsSuperieursOuEgaux ( Noeud<Integer> racine, int n )
```

19. Écrire une méthode qui prend en paramètre la référence vers le nœud racine d'un ABR ainsi qu'une chaîne de caractères. La méthode doit retourner le nombre de nœuds contenant une chaîne dont le préfixe (début de la chaîne) est la chaîne passée en paramètre. Tentez de faire une méthode efficace (il n'est pas nécessaire dans tous les cas de visiter chaque nœud de l'arbre). Voici l'entête de la méthode :

```
public static int nbNoeudsAvecPrefixe ( Noeud<String> racine, String prefixe )
```

20. Écrire une méthode qui prend en paramètre la référence vers le nœud racine d'un ABR contenant des chaînes de caractères. La méthode doit retourner true s'il s'agit d'un ABR, false sinon. Voici l'entête de la méthode :

```
public static boolean estABR ( Noeud<String> racine )
```

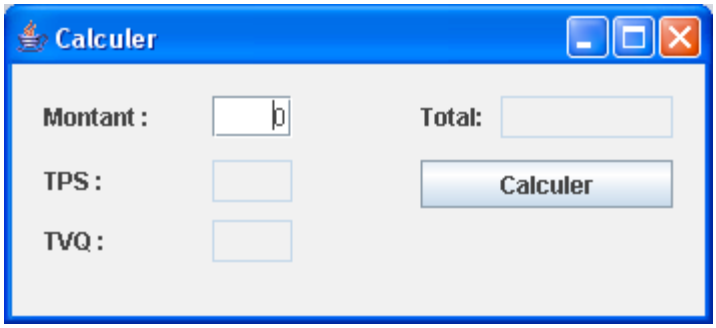
21. Écrire une méthode qui prend en paramètre la référence vers le nœud racine d'un ABR et qui crée une copie identique de cet arbre (il faut créer de nouveaux nœuds). La méthode retourne la référence vers le nœud racine de la copie. Voici l'entête de la méthode :

```
public static Noeud copie ( Noeud racine )
```

22. Écrire une méthode qui prend en paramètre une ArrayList contenant le parcours préfixe d'un ARB. La méthode retourne la référence vers le nœud racine de l'ABR qui a donné ce parcours préfixe. Voici l'entête de la méthode :

```
public static Noeud construire ( ArrayList parcoursPrefixe )
```

23. Vous devez écrire le code Java de la classe `Calculer` qui produira l'interface suivante :



L'interface permet d'entrer un montant d'achat et de calculer le montant de taxe de vente de la TPS, de la TVQ et le grand total. Le bouton `Calculer` déclenche le calcul des résultats.

Spécifications

1. L'utilisateur peut entrer des données seulement dans la case `montant`.
2. L'utilisateur peut taper la touche « enter » dans la case de saisie du montant pour déclencher le calcul ou en cliquant sur le bouton `Calculer`.
3. Le montant saisi doit être >= 0, sinon une boîte de dialogue (état erreur) sera affichée indiquant le message d'erreur.
4. La dimension de la fenêtre de l'application est de 350 x 160 pixels.
5. Les composants sont positionnés directement en spécifiant les coordonnées en pixels, donc l'interface n'utilise aucun gestionnaire de présentation.
6. Les cases TPS, TVQ et Total ne sont pas éditables.
7. La TPS est de 5% sur le montant. La TVQ est de 7.5% sur le montant incluant la TPS calculée.
8. La construction de l'interface se fera dans le constructeur de la classe `Calculer`.

La stratégie à utiliser pour le gestionnaire d'événements sera **`this.`**

24. L'exercice consiste à programmer l'interface Inventaire qui permet de saisir un nombre de caisses de pièces et un nombre de pièces par caisses. Le programme calculera le nombre de pièces au total lorsque le bouton Calculer sera cliqué ou lorsque l'utilisateur entrera <enter> dans l'un ou l'autre des deux champs de saisie. Le résultat sera affiché dans la case Total. Si l'usager entre un nombre négatif dans un des champs de saisie, une boîte de dialogue (JOptionPane) sera affichée pour informer l'utilisateur qu'on ne permet pas les valeurs < 0. Au départ, les valeurs 0 doivent être affichées dans chacun des trois champs.

```java
public class Maillon<T> {

    private T          info; // donnée dans le maillon
    private Maillon<T> suiv; // référence vers le maillon suivant

    /** Crée un nouveau maillon n'ayant pas de maillon suivant
     *  @param o l'information qui sera stockée dans le maillon
     */
    public Maillon ( T o ) {
        this ( o, null );
    }

    /** Crée un nouveau maillon ayant un autre maillon existant comme suivant
     *  @param o l'information qui sera stockée dans le maillon
     *  @param suivant le maillon qui sera le suivant du maillon créé
     */
    public Maillon ( T o, Maillon<T> suivant ) {
        info = o;
        suiv = suivant;
    }

    public T info () {
        return info;
    }

    public Maillon<T> suivant () {
        return suiv;
    }

    public void modifierInfo ( T o ) {
        info = o;
    }

    public void modifierSuivant ( Maillon<T> suivant ) {
        suiv = suivant;
    }
} // Maillon



public class Noeud<T> {

    private T info;          // information dans le noeud
    private Noeud<T> gauche; // sous-arbre gauche
    private Noeud<T> droite; // sous-arbre droit


    /** Cree un nouveau noeud feuille
     *  @param element l'information qui sera stockee dans le noeud
     */
    public Noeud ( T element ) {
        info = element;
    }

    /** Cree un nouveau noeud ayant un autre noeud existant a gauche et un autre noeud
     *  existant a droite
     *  @param element l'information qui sera stockee dans le noeud
     *  @param gauche le noeud qui sera le noeud a gauche du noeud cree
     *  @param droite le noeud qui sera le noeud a droite du noeud cree
     */
    public Noeud  ( T element, Noeud<T> gauche, Noeud<T> droite ) {
        info = element;
        this.gauche = gauche;
        this.droite = droite;
    }
```

```
    public T info () {
        return info;
    }

    public Noeud<T> gauche () {
        return gauche;
    }

    public Noeud<T> droite () {
        return droite;
    }

    public void modifierInfo ( T element ) {
        info = element;
    }

    public void modifierGauche ( Noeud<T> gauche ) {
        this.gauche = gauche;
    }

    public void modifierDroite ( Noeud<T> droite ) {
        this.droite = droite;
    }

} // Noeud
```