Identifiant de l'étudiant

Nom : Thomas Castonguay-Gagnon

Code permanent : CAST10059303

Consignes

1. Vous avez le droit à une seule remise. Seule la première remise sera corrigée, toute autre remise complète ou partielle (une ou plusieurs feuilles) ne sera pas considérée pour la correction.
2. Vous devez lire et respecter toutes les exigences générales relatives aux questionnaires. Ces exigences sont disponibles sur la page du cours.
3. Cochez la case suivante si vous avez lu et compris les exigences générales relatives aux questionnaires.

☑

1 Question

On vous demande de concevoir une classe `CarteMere` modélisant « une carte mère ». Pour chaque carte mère on souhaite conserver les informations suivantes (les attributs) :

- Sa marque (chaîne de caractères)
- Sa forme (on va seulement conserver un code entier). Voici les formes valides ainsi que le code associé (le code par défaut est 0) : 0 → "Inconnu", 1 → "ATX", 2 → "microATX", 3 → "flexATX", 4 → "miniATX", 5 → "miniITX", 6 → "nanoITX", 7 → "BTX", 8 → "microBTX", 9 → "picoBTX".
- La capacité mémoire maximale en Gigaoctets (un entier positif). La valeur par défaut est 8.
- La taille mémoire installée en Gigaoctets (un entier positif ou nul, inférieur ou égal à la capacité maximale). La valeur par défaut est 0.
- Un identifiant unique (nombre entier positif). La première carte créé doit avoir l'identifiant 1. La deuxième carte doit avoir l'identifiant 2, etc. À chaque nouvelle carte on incrément de 1 l'identifiant précédent.

Complétez la classe `CarteMere` suivante en respectant les spécifications indiquées dans les commentaires (sans dépasser le nombre de lignes vides disponibles) :

```
public class CarteMere {

    // Déclaration de constantes

    public static final int CAPACITE_MEMOIRE = 8;
    public static final String[] formes = {
        "Inconnue",
        "ATX",
        "microATX",
        "flexATX",
        "miniATX",
        "miniITX",
        "nanoITX",
        "BTX",
        "microBTX",
        "picoBTX"
    };


    // Variables d'instance (il faut les mettre private)
    private String marque;
    private int idForme;
    private int maxMem;
    private int memInstall;
    private int idCarte;









    // Variables de classe (il faut les mettre private)
    private static int idGlobal = 1;
```

```
48      /**
49       *  Ce constructeur crée une nouvelle carte ayant la marque marque,
50       *  un code forme codeForme (0 si codeForme n'est pas entre 0 et 9
51       *  inclusivement), une capacité mémoire capaciteMaxMemoire (8 si
52       *  capaciteMaxMemoire <= 0), une taille mémoire memoireInstallee
53       *  (0 si memoireInstallee < 0 et capaciteMaxMemoire si
54       *  memoireInstallee > capaciteMaxMemoire).
55       *  L'identifiant de la carte doit être calculé comme indiqué dans
56       *  l'énoncé.
57       *  @param marque Marque de la carte
58       *  @param codeForme Code entier de la forme de la carte
59       *  @param capaciteMaxMemoire capacité mémoire maximale
60       *  @param memoireInstallee taille de la mémoire installée sur la carte
61       */
62      public CarteMere (  String marque,
63                          int codeForme,
64                          int capaciteMaxMemoire,
65                          int memoireInstallee ) {
66          if (marque != null) {
67              this.marque = marque;
68          } else {
69              this.marque = "";
70          }
71          if (capaciteMaxMemoire > 0) {
72              maxMem = capaciteMaxMemoire;
73          } else {
74              maxMem = CAPACITE_MEMOIRE;
75          }
76          if (memoireInstallee < 0) {
77              memInstall = 0;
78          } else if (memoireInstallee > maxMem) {
79              memInstall = maxMem;
80          } else {
81              memInstall = memoireInstallee;
82          }
83          if (codeForme > 9 || codeForme < 0) {
84              idForme = 0;
85          } else {
86              idForme = codeForme;
87          }
88          idCarte = idGlobal;
89          idGlobal++;
90
91
92
93
94      }
95
```

```
96      /**
97       *  Ce constructeur crée une nouvelle carte comme le constructeur
98       *  précédent mais avec une marque correspondant à ASUS.
99       *  @param forme Code entier de la forme de la carte
100      *  @param capaciteMaxMemoire capacité mémoire maximale
101      *  @param memoireInstallee taille de la mémoire installée sur la carte
102      */
103     public CarteMere (  int forme, int capaciteMax, int memoireInstallee ) {
104
105         this("ASUS", forme, capaciteMax, memoireInstallee);
106
107
108
109
110
111
112
113
114
115
116
117
118
119     }
120
121     /**
122
123      *  Ce constructeur crée une nouvelle carte comme le constructeur
124      *  précédent mais avec une capacité mémoire maximale de 8Go.
125      *  @param codeForme Code entier de la forme de la carte
126      *  @param memoireInstallee taille de la mémoire installée sur la carte
127      */
128     public CarteMere (  int codeForme, int memoireInstallee ) {
129
130         this(codeForme, 0, memoireInstallee);
131
132
133
134
135
136
137
138
139
140
141
142     }
143
```

```
144     /**
145      * Retourne le numéro de la carte
146      * @return numéro de la carte
147      */
148     public int obtenirIdentifiant () {
149 
150         return idCarte;
151 
152 
153 
154     }
155 
156     /**
157      * Retourne la forme de la carte sous forme textuelle
158      * @return forme de la carte
159      */
160     public String obtenirForme () {
161 
162          return formes[idForme];
163 
164 
165 
166     }
167 
168     /**
169      * Retourne la capacité maximale de la mémoire
170      * @return capacité maximale de la mémoire
171      */
172     public int obtenirCapaciteMaxMemoire () {
173 
174          return maxMem;
175 
176 
177 
178     }
179 
180     /**
181      * Retourne la taille actuelle de la mémoire installée
182      * @return taille actuelle de la mémoire
183      */
184     public int obtenirMemoireInstallee () {
185 
186          return memInstall;
187 
188 
189 
190     }
191 
```

```
192     /**
193      * Modifie la forme de la carte si le paramètre est valide, sinon
194      * aucune modification ne sera apportée.
195      * @param nouvelleForme nouvelle forme
196      */
197     public void modifierForme ( int nouvelleForme ) {
198 
199         if (nouvelleForme >= 0 && nouvelleForme <= 9) {
200             idForme = nouvelleForme;
201         }
202 
203     }
204 
205     /**
206      * Modifie la marque en prenant la valeur reçue en paramètre.
207      * @param nouvelleMarque nouvelle marque
208      */
209     public void modifierMarque ( String nouvelleMarque ) {
210         if (nouvelleMarque != null) {
211             marque = nouvelleMarque;
212         }
213 
214 
215     }
216 
217     /**
218      * Modifie la taille de la mémoire installée sur la carte. Si la taille
219      * reçue est négative, alors il faut la forcer à 0. Si la taille reçue
220      * dépasse la capacité maximale alors il faut la forcer pour avoir la
221      * taille maximale possible. Dans les autres cas, il suffit d'utiliser
222      * la valeur reçue en paramètre.
223      * @param nouvelleTailleMemoire nouvelle taille
224      */
225     public void modifierMemoireInstallee ( int nouvelleTailleMemoire ) {
226         if (nouvelleTailleMemoire < 0) {
227             nouvelleTailleMemoire = memInstall;
228         } else if (nouvelleTailleMemoire > maxMem) {
229             nouvelleTailleMemoire = maxMem;
230         }
231         memInstall = nouvelleTailleMemoire;
232 
233 
234 
235 
236 
237 
238     }
239 
```

```
    /**
     *  Enlève toute la mémoire installée et retourne la taille de la
     *  mémoire qu'on vient d'enlever.
     *  @return taille mémoire enlevée
     */
    public int viderMemoire () {
        int mem = memInstall;
        memInstall = 0;
        return mem;









    }

    /**
     *  Ajoute de la mémoire jusqu'à la capacité maximale et retourne la
     *  taille de la mémoire qui a été ajoutée pour atteindre la capacité
     *  maximale.
     *  @return taille mémoire ajoutée pour atteindre la capacité maximale
     */
    public int installerMaxMemoire () {
        int add = 0;
        if (memInstall < maxMem) {
            add = maxMem - memInstall;
            memInstall = maxMem;
        }
        return add;









    }
```

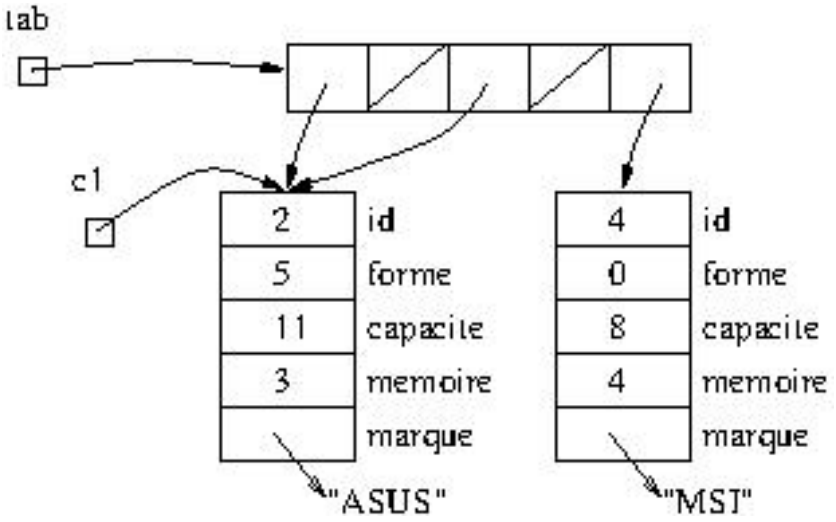
```
    /**
     *  Ajoute la taille memoireAdditionnelle à la taille actuellement
     *  installée selon les instructions suivantes.
     *  si memoireAdditionnelle < 0, alors aucune modification, et la méthode
     *  retourne tout simplement la valeur de memoireAdditionnelle.
     *  Si l'ajout de memoireAdditionnelle provoque un dépassement de la
     *  capacité maximale, il faudra alors ajouter seulement la quantité
     *  nécessaire pour atteindre la capacité maximale et la méthde doit
     *  alors retourner la quantité excédentaire.
     *  Dans les autres cas, la taille memoireAdditionnelle sera ajoutée
     *  à la mémoire installée et la méthode retournera la valeur 0.
     *  @param memoireAdditionnelle taille à ajouter
     *  @return memoireAdditionnelle si memoireAdditionnelle < 0,
     *          0 si pas de débordement,
     *          sinon la taille excédentaire.
     */
    public int ajouterMemoire ( int memoireAdditionnelle ) {
        int output = 0;
        if (memoireAdditionnelle < 0) {
            output = memoireAdditionnelle;
        } else if (memInstall + memoireAdditionnelle > maxMem) {
            output = memInstall + memoireAdditionnelle - maxMem;
            memInstall = maxMem;
        }else {
            memInstall = memoireAdditionnelle;
        }
        return output;
    }
```

```
336
337     /**
338      * Retourne une chaîne de caractères formée par:
339      * l'identifiant de la carte suivi par la forme sous forme textuelle
340      * suivie par la mémoire installée (séparés par des virgules).
341      * Par exemple, pour une carte 6Go de forme BTX ayant l'identifiant 2
342      * la méthode retourne: "2,BTX,6".
343      * @return chaîne qui décrit quelques attributs de la carte mère
344      */
345     public String toString() {
346         String output = "";
347         output = output + idCarte + ',' + this.obtenirForme() + ',' + memInstall;
348         return output;
349
350
351     }
352
353     /*
354      * Cette méthode reçoit en paramètre une forme de carte mère (sous forme
355      * textuelle) et doit retourner le code entier correspondant à cette
356      * forme.
357      * Dans le cas où la chaîne reçue en paramètre ne correspond à aucune
358      * forme valide, la méthode doit retourner le code 0.
359      * On supposera qu'aucun argument \texttt{null} ne sera passé en
360      * appelant cette méthode.
361      */
362     public static int chaineFormeVersCode ( String chaineForme ) {
363         int format = 0;
364             if (chaineForme.equalsIgnoreCase(formes[1])) {
365             for(int i = 1; i < formes.length ; i++){
366                 if (chaineForme.equalsIgnoreCase(formes[i]){
367                     format = i;
368                 }
369             }
370         return format;
371
372
373
374
375
376
377
378     }
379
380 }
```

2 Question

Remarque : Dans la représentation de la mémoire on ignore les variables de classe.

On vous demande d'écrire un programme qui utilise la classe `CarteMere` demandée dans la question précédente et qui permet d'obtenir la représentation de la mémoire suivante (ne pas dépasser le nombre de lignes vides et ne pas déclarer plus que deux variables) :



Votre réponse :

```
public class TesterCarteMere {
    public static void main ( String [] args ) {
        CarteMere[] tab = new CarteMere[5];
        CarteMere c1 = new CarteMere(0, 0);
        tab[0] = new CarteMere("ASUS", 5, 11, 3);
        c1 = new CarteMere(0, 0);
        tab[2] = tab[0];
        c1 = tab[0];
        tab[4] = new CarteMere("MSI", 0, 8, 4);









    }
}
```

3 Question

En vous basant sur la représentation de mémoire de la question précédente exécutez les instructions suivantes dans l'ordre sur un brouillon et dessinez la nouvelle représentation de la mémoire après exécution de ces instructions (votre réponse à la question doit correspondre à la représentation de la mémoire après exécution de ces instructions en partant de la question précédente) :

1. `c1 = tab[4];`
2. `tab[3] = c1;`
3. `tab[2] = tab[0];`
4. `tab[0].modifierForme(7);`
5. `tab[2].modifierMemoireInstallee(16);`
6. `tab[3].ajouterMemoire(2);`
7. `tab[4].viderMemoire();`

Votre réponse :