



LU2IN002-2022oct  
Éléments de programmation  
par objets avec Java

Contrôle du 9 novembre 2022 – Durée : 1 heure 30 minutes

Documents non autorisés. Appareils électroniques éteints et rangés.  
Le barème sur 40 points est donné à titre indicatif.

## Partie 1 Exercices (20 points)

**Exercice 1 (3 points)** Soit la classe **A** et les instructions suivantes.

```

1 public class A {
2     public A getNew() {
3         return new A();
4     }
5 }
6 // main
7 A a1=new A();
8 A a2=a1.getNew();
9 A a3=a1;
10 A a4=a2;
11 A [] tab=new A[5];
12 tab[0]=a1;
13 tab[1]=a2;
14 tab[2]=a4;
15 tab[3]=new A();

```

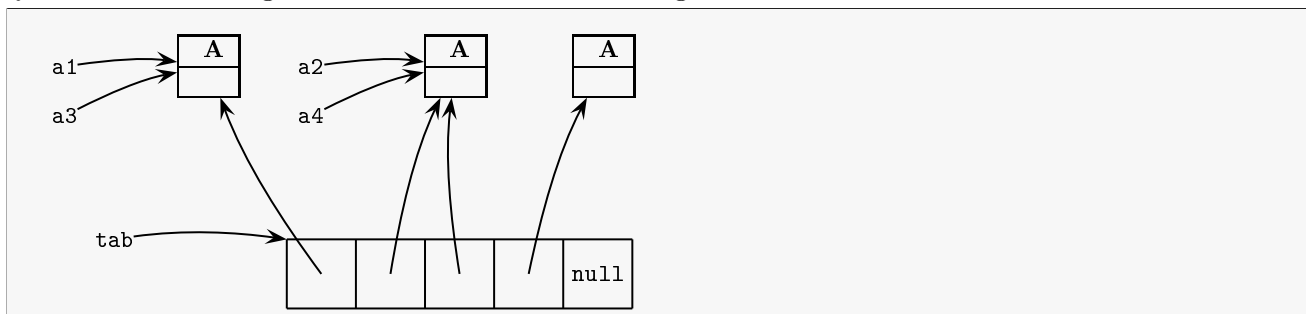
**Q1.1** Entre les lignes 7 et 15 : **(a)** combien d'instances de la classe **A** ont été créées ? **(b)** combien de variables au total ont été déclarées (y compris les variables de type tableau) ?

**(a)** 3 instances : lignes 7, 8 (3) et 15

**(b)** Il y a 5 variables nommées (**a1**, **a2**, **a3**, **a4** et **tab**). On peut aussi compter comme variable les 5 cases mémoires du tableau, cela fait donc au total 10 variables.

Remarque : la méthode **getNew()** pourrait être static.

**Q1.2** Dessiner le diagramme mémoire à la fin de la ligne 15.



**Exercice 2 (8 points)** On considère des carreaux colorés qui contiennent une lettre.

```

1 public class Carreau {
2     private char lettre;
3     private String couleur;
4     public Carreau(char let, String coul){
5         lettre=let; couleur=coul;
6     }
7     public String toString() {
8         return lettre+"-"+couleur;
9     }
10 }

```

ainsi que les instructions suivantes :

```

11 Carreau c1=new Carreau('A',"gris");
12 Carreau c2=new Carreau('C',"marron");
13 Carreau c3=c1.addition(c2);
14 System.out.println("c3="+c3);

```

Affiche : "c3=B-grismarron"

**Q2.1** Écrire la méthode **addition** de la classe **Carreau** telle que l'addition entre un carreau et un autre carreau retourne un nouveau carreau qui a pour valeur :

— pour la variable d'instance **lettre** : la partie entière de la moyenne des codes ASCII des lettres des deux carreaux. Exemples :  $\lfloor ('A'+'C')/2 \rfloor = 'B'$ ,  $\lfloor ('A'+'D')/2 \rfloor = 'B'$ ,  $\lfloor ('A'+'E')/2 \rfloor = 'C'$ ...

Remarque : la partie entière peut être obtenue avec un cast.

— pour la variable d'instance **couleur** : la concaténation des deux couleurs.

Voir exemple d'utilisation et d'affichage aux lignes 13 et 14.

```
public Carreau addition(Carreau c2) {
    return new Carreau((char)((lettre+c2.lettre)/2),couleur+c2.couleur);
}
```

**Q2.2** On souhaite écrire une classe **Usine** pour créer des carreaux. Donner le code de cette classe avec :

- un (seul) attribut **tabCoul** de type tableau de chaînes de caractères initialisés avec des noms de couleurs, par exemple : blanc, gris, marron,
- un (seul) constructeur privé qui ne fait rien,
- une (seule) méthode **getCarreau** qui prend en paramètre une lettre (type **char**) et qui retourne un objet de type **Carreau**. La couleur du carreau est initialisée aléatoirement à partir des couleurs du tableau.

```
public class Usine {
    private static String [] tabCoul={"blanc", "gris", "marron"};
    private Usine() {}
    public static Carreau getCarreau(char lettre) {
        int alea=(int)(Math.random()*tabCoul.length);
        return new Carreau(lettre,tabCoul[alea]);
    }
}
```

**Q2.3** On représente une phrase par un tableau à 2 dimensions de caractères, où chaque ligne représente un mot. Par exemple, la phrase : "Bonjour à tous" peut être représentée par le schéma ci-contre, ce qui correspond à la déclaration Java suivante :

B	o	n	j	o	u	r
à						
t	o	u	s			

```
char [][] phrase={{'B','o','n','j','o','u','r'},{'à'},{'t','o','u','s'}};
```

On cherche à réaliser un carrelage mural (tableau de carreaux pour mettre au mur) correspondant à une phrase quelconque donnée. On suppose que l'on est dans une méthode **main** d'une classe **Test** et qu'une variable **phrase** de type **char[][]** a été déclarée et initialisée avec une phrase quelconque. Écrire les instructions pour réaliser les opérations suivantes :

- Déclarer un tableau **tabC** à 2 dimensions de **Carreau** ayant exactement les mêmes dimensions que le tableau **phrase**.
- Pour chaque case du tableau **phrase**, demander à l'usine un nouveau carreau et l'affecter à la case correspondante dans le tableau **tabC**.

```
Carreau [][] tabC=new Carreau[phrase.length][];
for(int i=0; i < phrase.length; i++) {
    tabC[i]=new Carreau[phrase[i].length];
    for(int j=0; j < tabC[i].length; j++) {
        tabC[i][j]=Usine.getCarreau(phrase[i][j]);
    }
}
```

**Exercice 3 (9 points)** Soit la classe **Resto** ci-dessous qui compile sans erreur.

```
1 public class Resto {
2     private static int cpt=0;
3     protected final int idR;
4     protected String type;
5     public final String nom;
6     public static final String LETTRES="Resto";
7
8     public Resto(String type) {
9         cpt++;
10        idR=cpt;
11        this.type=type;
12        this.nom=LETTRES+idR;
13    }
14
15    public static int getCpt() {
16        return cpt;
17    }
18
19    public void resetCpt() {
20        cpt=0;
21    }
22
23    public String toString() {
24        return nom+" "+type;
25    }
26 }
```

**Q3.1** Pour chaque instruction des lignes 31 à 34 et 41 à 44 (8 instructions), écrire le numéro de ligne,

puis indiquer si l'instruction compile (OK) ou pas (FAUX). Si l'instruction ne compile pas, justifiez en quelques mots pourquoi (exemples de justification : privé, final, pas statique...).

```

30 Resto r1=new Resto("classique");
31 System.out.println(r1.cpt);
32 System.out.println(r1.nom);
33 System.out.println(r1.LETTRES);
34 System.out.println(r1.getCpt());
41 System.out.println(Resto.cpt);
42 System.out.println(Resto.nom);
43 System.out.println(Resto.LETTRES);
44 Resto.resetCpt();

```

1.31 r1.cpt : FAUX cpt privé	1.41 Resto.cpt : FAUX cpt privé
1.32 r1.nom : OK	1.42 Resto.nom : FAUX nom pas <b>static</b>
1.33 r1.LETTRES : OK	1.43 Resto.LETTRES : OK
1.34 r1.getCpt() : OK	1.44 Resto.resetCpt() : FAUX méthode pas <b>static</b>

Remarque : la méthode `resetCpt()` devrait être `static`, car elle ne dépend pas d'une instance, ce qui permettrait dans le main de faire : `Resto.resetCpt()`.

**Q3.2** Dans la classe `Resto`, on ajoute la méthode ci-dessous. Est-ce que la classe compile toujours ? Expliquez brièvement.

```

public static void afficherInfo() {
    System.out.println(nom+" "+type);
}

```

Non, on ne peut pas utiliser de variables d'instance dans une méthode `static`. Cette méthode qui dépend de l'instance ne devrait pas être `static`.

```

public void afficherInfo() { // Bonne solution
    System.out.println(nom+" "+type);
}

```

**Q3.3** On considère maintenant la classe `FastFood` suivante. La méthode `modifierInfo()` compile avec des erreurs. Pour chaque instruction des lignes 62 à 65 (4 instructions), écrire le numéro de ligne, puis indiquer si l'instruction compile (OK) ou pas (FAUX). Si l'instruction ne compile pas, justifiez en quelques mots pourquoi (exemples de justification : privé, final, pas statique...).

```

51 public class FastFood extends Resto {
52     private int prixMax;
53     public FastFood(int prixMax) {
54         super("FastFood");
55         this.prixMax=prixMax;
56     }
57     public String toString() {
58         return super.toString()
59             +" prixMax="+prixMax;
60     }
61     public void modifierInfo() {
62         cpt++;
63         idR=100;
64         type="FF";
65         nom="FF100";
66     }
67 }

```

1.62 cpt++; // FAUX privé
1.63 idR=100; // FAUX final
1.64 type="FF"; // OK
1.65 nom="FF100"; // FAUX final

**Q3.4** Soit les instructions ci-dessous. Sachant que `r1.toString()` retourne `"Resto1 classique"`, qu'affiche exactement les lignes 5 et 6 ?

```

1 Resto r1=new Resto("classique");
2 FastFood ff1=new FastFood(15);
3 Resto r2=new FastFood(25);
4 System.out.println(r1.toString());
5 System.out.println(ff1.toString());
6 System.out.println(r2.toString());

```

1.5 : "Resto2 FastFood prixMax=15"  
 1.6 : "Resto3 FastFood prixMax=25" (c'est le `toString` du type de l'objet qui est appelé)

## Partie 2 Problème (25 points)

**Remarque préliminaire :** la visibilité des variables et des méthodes à définir, ainsi que leurs aspects statique, ou final ne sera pas précisé. C'est à vous de décider de la meilleure solution à apporter. Si la méthode `toString` n'est pas demandée dans la question, il n'est pas nécessaire de la fournir. Sans aucune précision donnée dans la question, une méthode ne rend rien.

Nous souhaitons écrire un programme pour la gestion des articles scientifiques soumis à une conférence internationale et qui doivent constituer les actes de cette conférence. De tels actes sont constitués d'articles. Un article possède un titre, une liste d'auteurs, ainsi qu'un identifiant unique.

**Question 4.1 (3.5 points)** Écrire la classe `Article`. Cette classe doit comporter un unique constructeur prenant en argument une chaîne de caractères `titre` correspondant au titre de l'article. Chaque article doit posséder un identifiant entier `ident` unique que l'on doit pouvoir consulter (mais pas de le modifier) sans passer par un accesseur. Dans cette classe, la liste des auteurs est représentée par un tableau de chaînes de caractères, qui peut comporter au plus 10 auteurs. La classe possède la méthode `ajouteAuteur(String auteur)` qui permet d'ajouter un auteur à la liste si la limite du nombre d'auteurs n'a pas été atteinte (il ne se passe rien dans le cas contraire). La classe est complétée par un accesseur en lecture pour le titre de l'article, et un autre accesseur pour son nombre d'auteurs. Une méthode `toString()` permet de récupérer l'ensemble des informations liées à l'article.

```
1 public class Article {
2     private static int compteur = 0;
3     public final int ident;
4     private String titre;
5     private String[] auteurs;
6     private int nbAuteurs = 0;
7
8     public Article(String titre) {
9         this.ident = compteur++;
10        this.titre = titre;
11        auteurs = new String[10]; // un maximum de 10 auteurs
12    }
13
14    public String getTitre() { return titre; }
15    public int getNbAuteurs() { return nbAuteurs; }
16
17    public void ajouteAuteur(String a) {
18        if (nbAuteurs < auteurs.length) {
19            auteurs[nbAuteurs++] = a;
20        }
21    }
22
23    @Override
24    public String toString() {
25        String res = "[" + this.ident + ": ";
26        res += this.titre;
27        for (int i=0; i<this.nbAuteurs;i++)
28            res += ", " + auteurs[i];
29        if (nbAuteurs == 0)
30            res += ", pas d'auteurs";
31
32        return res;
33    }
34 }
```

**Question 4.2 (2 points)** Les articles peuvent être de 2 types : les articles longs qui ont un nombre de pages et les articles courts qui eux présentent des travaux appliqués soit industriels, soit académiques.

1. Écrire la classe **ArticleLong** qui spécialise la classe **Article**. Dans cette classe, le nombre de pages est un entier auquel un accesseur (getter) permet d'accéder. Cette classe comporte 2 constructeurs : un constructeur prenant en argument le titre et le nombre de pages, et un constructeur pour créer des articles de titre donné et qui font 8 pages. Une méthode **toString()** est aussi fournie.

```

1  public class ArticleLong extends Article {
2      private int nbPages;
3
4      public ArticleLong(String titre , int nbPages) {
5          super(titre);
6          this.nbPages = nbPages;
7      }
8
9      public ArticleLong(String titre) {
10         this(titre,8);
11     }
12
13     public int getNbPages() { return nbPages; }
14
15     @Override
16     public String toString() {
17         return "Article long (" + nbPages + " p): " + super.toString();
18     }
19 }

```

2. De la même façon, écrire la classe **ArticleCourt** qui possède un attribut booléen associé au type des travaux présentés dans l'article (industriels ou académiques). Cette classe est complétée par un constructeur prenant en argument le titre de l'article et le type des travaux (booléen vrai si "industriel"), ainsi que par une méthode **toString()**.

```

1  public class ArticleCourt extends Article {
2
3      private boolean industriel;
4      public ArticleCourt(String titre , boolean industriel) {
5          super(titre);
6          this.industriel = industriel;
7      }
8      @Override
9      public String toString() {
10         return "Article court " + (industriel ? "(industriel) " : "(académique) ") + super.toString();
11     }

```

**Question 4.3 (2 points)** Écrire la classe **Test** qui ne comporte qu'une méthode **main()** dans laquelle les étapes suivantes sont implémentées :

- création d'un tableau de 5 articles dont le nom est **tabArticles**;
- en première position dans **tabArticles**, création d'un article long de titre "The Java Language" de 8 pages et ajout des 2 auteurs "Gosling" et "Naughton" à cet article;
- en deuxième position dans **tabArticles**, création d'un article long de titre "Artificial Intelligence" de 42 pages et ajout des 2 auteurs "McCarthy", "Minsky" et "Simon" à cet article;
- en troisième position dans **tabArticles**, création d'un article court de titre "Python applied" de type applications industrielles et ajout de l'auteur "Van Rossum" à cet article;
- affichage des informations sur les articles du tableau **tabArticles** sous la forme ci-dessous :  
Article 1: Article long (8 p): [0]: The Java Language, Gosling, Naughton  
Article 2: Article long (42 p): [1]: Artificial Intelligence, McCarthy, Minsky, Simon  
Article 3: Article court (industriel) [2]: Python, Van Rossum

Remarque : il y a 2 bugs dans l'intitulé de la question :

- dans le 3e item, il faut lire "de 42 pages et ajout de 3 auteurs "McCarthy", "Minsky" et

"Simon" (et pas 2 auteurs, ...).

— il est affiché "Python, Van Rossum" pour l'article 3, selon le texte de la question, cela devrait être "Python applied, Van Rossum".

```

1 public class Test {
2     public static void main(String[] args) {
3         Article[] tabArticles = new Article[5];
4
5         tabArticles[0] = new ArticleLong("The Java Language");
6         tabArticles[0].ajouteAuteur("Gosling");
7         tabArticles[0].ajouteAuteur("Naughton");
8
9         tabArticles[1] = new ArticleLong("Artificial Intelligence",42);
10        tabArticles[1].ajouteAuteur("McCarthy");
11        tabArticles[1].ajouteAuteur("Minsky");
12        tabArticles[1].ajouteAuteur("Simon");
13
14        tabArticles[2] = new ArticleCourt("Python",true);
15        tabArticles[2].ajouteAuteur("Van Rossum");
16
17        for (int i=0; i<tabArticles.length; i++) {
18            if (tabArticles[i] != null) {
19                System.out.println("Article "+(i+1)+": "+
20                                   tabArticles[i]);
21                // Rem: parenthèses obligatoires pour i+1 !
22            }
23        }
24    }

```

**Question 4.4 (4 points)** Lors d'une conférence, tous les articles doivent donner lieu à une présentation par leurs auteurs : les articles longs sont présentés lors d'un exposé, les articles courts sont présentés sous forme de poster. Dans notre application, la présentation correspond à l'affichage d'un message composé par "Exposé sur" ou par "Poster sur", selon le cas, et complété par le titre de l'article. Par exemple :

Exposé sur The Java Language

Exposé sur Artificial Intelligence

Poster sur Python

1. On souhaite ajouter la méthode `presentation()` qui ne rend rien mais affiche le message correspondant à l'article présenté. A quel(s) niveau(x) cette méthode doit-elle être introduite ? Proposer le code correspondant à son bon fonctionnement.

La méthode doit être ajoutée dans les 3 classes `Article`, `ArticleLong` et `ArticleCourt`.

dans `Article` :

```

1 public void presentation() {}

```

dans `ArticleLong` :

```

1 public void presentation() {
2     System.out.println("Exposé sur "+getTitre());
3 }

```

dans `ArticleCourt` :

```

1 public void presentation() {
2     System.out.println("Poster sur "+getTitre());
3 }

```

2. Donner les instructions à ajouter dans `main()` (à la suite du code écrit dans la question 4.3) pour afficher le message donné en exemple ci-dessus.

```

Dans le main() :
1  for (Article a : tabArticles) {
2      if (a != null) {
3          a.presentation();
4      }
5  }
ou autre solution possible :
1  for (int i=0; i< tabArticles.length; i++) {
2      if (tabArticles[i] != null)
3          tabArticles[i].presentation();
4  }

```

3. La solution actuelle n'est pas optimale car elle n'empêche pas la création d'article ni long, ni court, ce qui peut poser problème pour la présentation. Proposer une modification de la classe **Article** pour rendre plus propre notre solution.

C'est une question de cours qui porte sur le cours 7 (pas vu en TD) : la bonne solution est de rendre abstraite la classe **Article** et la méthode **presentation()** dans cette classe.

**Question 4.5 (1.5 points)** Écrire un constructeur par recopie pour la classe **Article** qui permette de copier un article existant et telle que la copie obtenue possède un identifiant différent.

```

1  public Article(Article art) {
2      this.titre = art.titre;
3      this.auteurs = new String[10];
4      this.nbAuteurs = art.nbAuteurs;
5      for (int i=0; i<this.nbAuteurs; i++)
6          this.auteurs[i] = art.auteurs[i];
7  }

```

**Question 4.6 (1 point)** Écrire un constructeur par recopie pour les classes **ArticleLong** et **ArticleCourt** selon le modèle de la question précédente.

```

— dans ArticleLong :
1      public ArticleLong(ArticleLong art) {
2          super(art);
3          this.nbPages = art.nbPages;
4      }
— dans ArticleCourt :
1      public ArticleCourt(ArticleCourt art) {
2          super(art);
3          this.industriel = art.industriel;
4      }

```

**Question 4.7 (1 point)** Donner les instructions à ajouter dans **main()** pour ajouter une copie des 2e et 3e articles dans **tabArticles** à la suite des articles existants.

```

Dans le main() : ne pas oublier le cast !
1  tabArticles[3] = new ArticleLong((ArticleLong)tabArticles[1]);
2  tabArticles[4] = new ArticleCourt((ArticleCourt)tabArticles[2]);

```

**Question 4.8 (3 points)** Lors d'une conférence, l'ensemble des articles longs qui y sont présentés sont regroupés en un livre qui porte le nom d'"actes" de la conférences. On souhaite définir une classe **Actes** pour représenter ce document.

Écrire la classe **Actes** qui possède comme attributs un **titre** (chaîne de caractères) et une liste de d'articles longs définie sous la forme d'une **ArrayList** d'**ArticleLong**. Cette classe doit posséder un constructeur à 1 argument : une chaîne de caractères donnant le titre des actes (par exemple "conférence IJCAI 2022"). La classe doit aussi contenir les méthodes suivantes : une méthode **ajouteArticle** permettant d'ajouter un **ArticleLong** à la liste des articles, une méthode **getNbPages** pour obtenir le nombre de pages

total des articles qu'elle contient, une méthode `getNbArticles` donnant le nombre d'articles contenu dans sa liste, et une méthode `toString` donnant l'ensemble des informations (titre, nombre de pages, et liste des articles inclus).

```

1 import java.util.ArrayList;
2
3 public class Actes {
4     private String titre;
5     private ArrayList<ArticleLong> liste;
6
7     public Actes(String titre) {
8         this.titre = titre;
9         liste = new ArrayList<ArticleLong>();
10    }
11
12    public void ajouteArticle(ArticleLong art) {
13        liste.add(art);
14    }
15    public int getTotalPages() {
16        int totalPages = 0;
17        for (ArticleLong art : liste) {
18            totalPages += art.getNbPages();
19        }
20        return totalPages;
21    }
22
23    public int getNbArticles() { // utilisation obligatoire de size()
24        return liste.size();
25    }
26    @Override
27    public String toString() {
28        String res = "Actes " + titre + " (" + getTotalPages() + "p):\n";
29        for (int i=0; i<liste.size(); i++) {
30            res += "\t" + liste.get(i) + "\n";
31        }
32        return res;
33    }
34 }

```

Une autre solution acceptable pour `toString()` :

```

1 public String toString2() {
2     String res = "Actes " + titre + " (" + getTotalPages() + "p):\n";
3     for (ArticleLong art : liste) {
4         res += "\t" + art + "\n";
5     }
6     return res;
7 }

```

**Question 4.9 (2 points)** On souhaite donner des arguments lors du lancement du programme Java que nous venons d'écrire de la façon suivante :

- si aucun argument n'est donné, rien de particulier à faire ; par exemple si on lance :  
`java Test`
- si un argument est donné, alors créer un article court dont l'argument donné est le titre et qui est de type académique ; par exemple si on lance :  
`java Test MrPython`
- si deux arguments sont donnés, alors créer un article long dont le premier argument est le titre et le second argument est le nombre de pages ; par exemple si on lance :  
`java Test "Artificial Intelligence" 42`

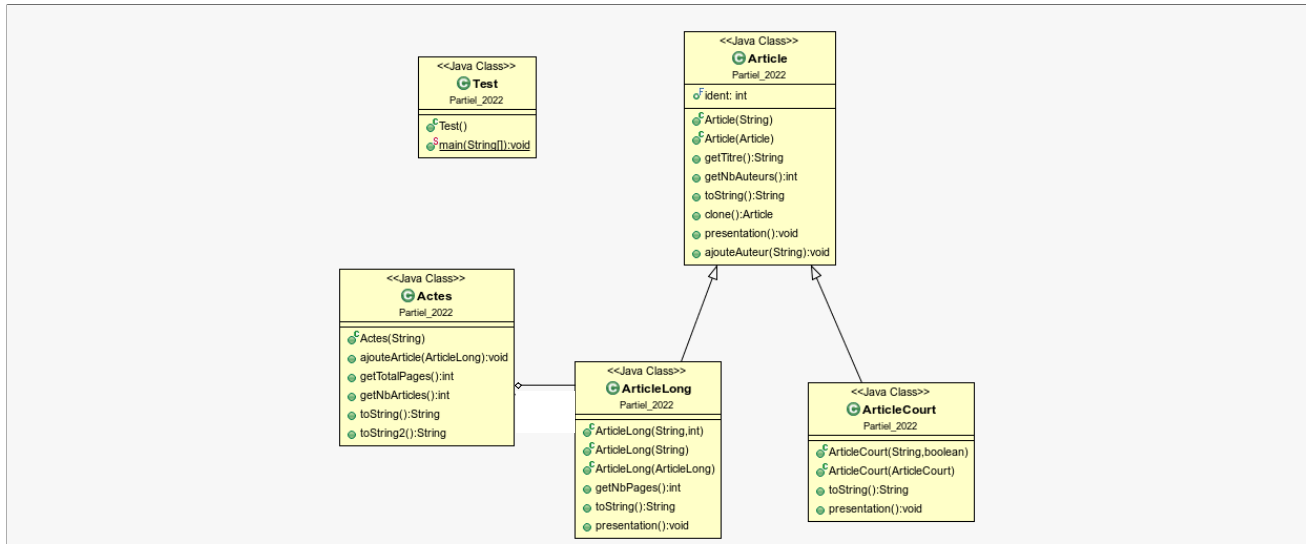
Donner les instructions à ajouter dans la fonction `main` de la classe `Test` pour réaliser cela.

```

1  System.out.println("\nUtilisation des args: ");
2  Article artCree = null;
3  if (args.length == 0) {
4      System.out.println("Pas de création");
5  }
6  else if (args.length == 1)
7      artCree = new ArticleCourt(args[0], false);
8  else
9      artCree = new ArticleLong(args[0], Integer.parseInt(args[1]));
10
11 System.out.println(artCree);

```

**Question 4.10 (1.5 points)** Donner le schéma UML **client** avec toutes les classes réalisées.



**Question 4.11 (4 points)** Pour garantir l'unicité des actes d'une conférence, il est nécessaire de proposer une classe qui ne puisse pas être instanciée plus d'une fois. Donner la définition de la classe **ActesUniques** qui spécialise la classe **Actes** et dont on peut garantir l'existence d'une unique instance lors de l'exécution du programme. Le titre pour des actes uniques doit obligatoirement être "LU2IN002".

Donner ensuite les instructions à ajouter dans le **main** précédent afin de créer une instance d'**ActesUniques**, de lui ajouter les 2 premiers articles de la question 4.3, puis d'afficher son contenu.

Définition de la classe, on peut utiliser une variable public final pour éviter de d'écrire une méthode statique :

```

1 public class ActesUniques extends Actes {
2     public static final ActesUniques INSTANCE = new ActesUniques();
3     private ActesUniques() {
4         super("LU2IN002");
5     }
6 }

```

ou on peut utiliser une méthode statique pour récupérer l'instance :

```

1 public class ActesUniquesV2 extends Actes {
2     private static final ActesUniquesV2 INSTANCE = new ActesUniquesV2();
3     private ActesUniquesV2() {
4         super("LU2IN002");
5     }
6     public static ActesUniquesV2 getInstance() {
7         return INSTANCE;
8     }
9 }

```

Utilisation de la 1ère solution :

```
ActesUniques actesUn = ActesUniques.INSTANCE;  
actesUn.ajouteArticle((ArticleLong)tabArticles[1]);  
actesUn.ajouteArticle((ArticleLong)tabArticles[0]);  
System.out.println(actesUn);
```

Utilisation de la 2e solution :

```
ActesUniquesV2 actesUnV2 = ActesUniquesV2.getInstance();  
actesUnV2.ajouteArticle((ArticleLong)tabArticles[1]);  
actesUnV2.ajouteArticle((ArticleLong)tabArticles[0]);  
System.out.println(actesUnV2);
```