

Bruno Bouzy

[bouzy@math-info.univ-paris5.fr](mailto:bouzy@math-info.univ-paris5.fr)

18 octobre 2002

Ce document est un sujet de TP de Java qui peut être associé ou bien à un cours de Java ou à cours de Génie Logiciel. Il a été mis au point suite aux enseignements de Java et Génie Logiciel donné en MIAIF2, MST2, IASV à l'Ufr de Mathématiques et Informatique de l'université Paris 5 entre 1997 et aujourd'hui.

Dans le cadre d'un module de Génie Logiciel l'objectif visé est :

- l'écriture d'un programme simple pour un utilisateur,
- la programmation d'un modèle UML très simple,
- la mise en œuvre d'associations entre instances.

et l'objectif visé n'est pas :

- l'utilisation d'outils de gestion de base de données.

Dans le cadre d'un cours de Java, l'objectif visé est la maîtrise de concepts Java suivants :

- contenu d'une classe, constructeur, références,
- propriété de classe (mot-clé `static`) ou propriété d'instance,
- la classe `String` et la classe `Vector`,
- chargement et sauvegarde sur fichiers.

et les points suivants ne sont pas abordés :

- visibilité des propriétés, encapsulation des données,
- généralisation, héritage, redéfinition,
- utilisation d'un environnement de programmation Java.

Prérequis :

- connaissance du C ou C++, de UML éventuellement,
- un compilateur Java, ici `javac`, un interpréteur Java, ici `java` du JDK.

Ce document contient :

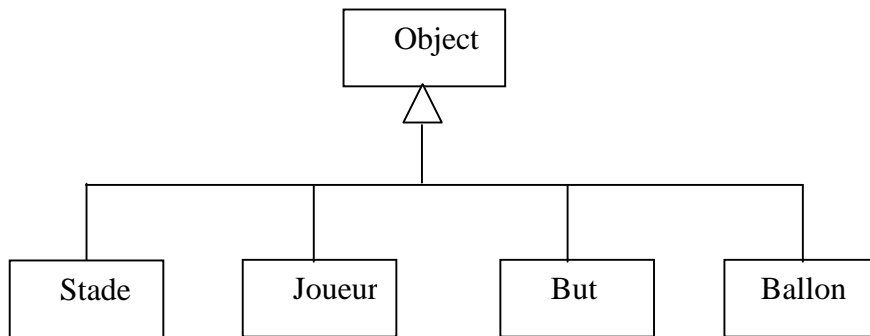
- un bref modèle UML du programme final `stade`,
- un TP « premiers pas »,
- un TP « créer détruire des instances avec interaction utilisateur »,
- un TP « associer dissocier des instances avec interaction utilisateur »,
- un TP « charger sauver des instances sur fichier ».

**Introduction**

Ce document est le résumé des Spécifications des Besoins du Logiciel STADE.

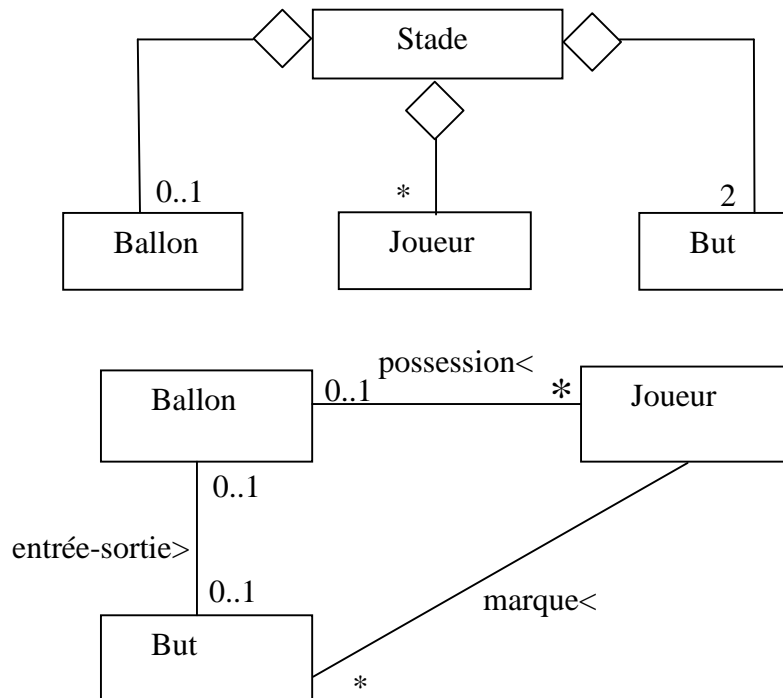
**Rappels : les diagrammes globaux du logiciel STADE**

Diagramme de généralisation :



(nb : ce diagramme est donné à titre de complétude car la généralisation, ici en râteau, est simplissime)

Diagrammes de classes



(nb : un joueur peut posséder le ballon qui peut être possédé par plusieurs joueurs (une mêlée ?), un ballon peut être dans un but qui contient éventuellement un ballon, un joueur peut marquer entre 0 et plusieurs buts, un but est marqué par un joueur exactement).

### Créer une mini-application avec une seule classe

Le but de ce paragraphe est d'être capable d'écrire un programme qui saisit des informations et les affiche. Créer un fichier `stade.java` et écrire le contenu suivant dans le fichier `stade.java` :

```
class Stade {
    public static void main (String args[])
    {
        System.out.println("Projet stade");
    }
}
```

Pour afficher des informations à l'écran, Java dispose de la méthode `println()` de la classe `Out`. Que voyez-vous à l'exécution ?

Insérer le fichier `Keyboard.java` dans votre projet et rajouter la méthode `Keyboard.pause()` afin d'éviter la disparition de la fenêtre d'exécution :

```
public static void main (String args[])
{
    System.out.println("Projet stade");
    Keyboard.pause();
}
```

Que voyez-vous à l'exécution ?

### Saisir des informations sur l'entrée standard

Pour saisir des informations au clavier, Java dispose de la classe `In`. Mais celle-ci est compliquée pour être utilisée telle quelle. Modifier la méthode `main()` comme suit :

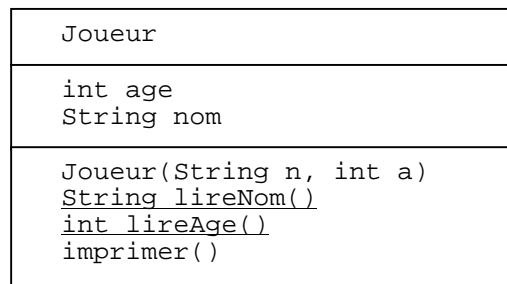
```
class Stade {
    public static void main (String args[])
    {
        System.out.println("Projet stade");
        System.out.println("Taper une chaine de caracteres:");
        String s = Keyboard.getString();
        System.out.println("La chaine lue est:\n<"+s+">");
        System.out.println("Taper un entier:");
        int i = Keyboard.getInt();
        System.out.println("Le double de l'entier lu est:" + 2*i);
        Keyboard.pause();
    }
}
```

Maintenant vous savez faire un programme qui lit une chaîne de caractères ou un entier sur l'entrée standard et qui affiche une chaîne de caractères sur la sortie standard.

### Créer une mini-application avec 2 classes

Le but de ce paragraphe est d'être capable d'écrire un programme qui comprend 2 classes : une classe utilisatrice, la classe `Stade`, et une classe utilisée, la classe `Joueur`. La classe utilisatrice lit des informations sur les propriétés d'une instance à créer ; elle

créé l'instance avec ses propriétés ; elle imprime l'instance sur la sortie standard. On suppose que la classe `Joueur` est définie comme suit :



`Joueur(String n, int a)` est un constructeur avec 2 arguments, le premier pour valoriser l'attribut `nom` et le second pour valoriser l'attribut `age`.

`String lireNom()` est une méthode de classe qui demande à l'utilisateur de taper un nom de joueur au clavier et qui retourne cette chaîne. C'est une méthode quasi-identique à `Keyboard.getString()` ; ne pas confondre avec la méthode d'instance d'accès à l'attribut `nom` quand celui-ci est privé. Tout au long de ces TPs, nous n'utiliserons pas de méthode d'accès aux attributs ; tous les attributs et toutes les méthodes sont *publiques*.

`int lireAge()` est une méthode de classe qui demande à l'utilisateur de taper un âge de joueur au clavier et qui retourne cette chaîne. C'est une méthode quasi-identique à `Keyboard.getInt()` ; ne pas confondre avec la méthode d'instance d'accès à l'attribut `age` quand celui-ci est privé.

`imprimer()` est une méthode d'instance qui affiche les attributs du joueur à l'écran.

Créer un fichier `Joueur.java` qui correspond à la spécification UML ci-dessus.

Afin de tester la classe `Joueur`, modifier le programme principal comme suit :

```
public static void main (String args[])
{
    System.out.println("Projet stade");
    String s = Joueur.lireNom();
    int a = Joueur.lireAge();
    System.out.println("s = <" + s + ">");
    System.out.println("a = <" + a + ">");
    Joueur j = new Joueur(s, a);
    System.out.println("Joueur j cree.");
    j.imprimer();
    Keyboard.pause();
}
```

Remarquer que le programme demande d'abord le nom et l'âge du joueur à l'utilisateur, puis connaissant les valeurs du nom et de l'âge, il crée une instance en appelant le constructeur de la classe `Joueur` avec ces valeurs. Remarquer que lors de l'appel de `lireNom` et `lireAge`, aucune instance de joueur n'existe. Elles sont des méthodes de classe et pas des méthodes d'instance.

Maintenant vous savez faire un programme qui lit des propriétés d'une instance d'une classe, qui crée l'instance et qui l'affiche.

# TP JAVA : CREER DETRUIRE INSTANCE AVEC INTERACTION UTILISATEUR

---

## Introduction

Ce TP reprend la suite du TP dans lequel le programme principal permettait de créer un joueur en demandant à l'utilisateur de taper le nom et l'âge du joueur au clavier. Ici, nous remplaçons un joueur par *des ballons*. Nous allons créer une classe `Ballon` avec laquelle l'utilisateur pourra non seulement *créer* des ballons identifiés par une `String`, mais aussi les *afficher* et les *détruire*. Il va donc être nécessaire d'avoir un menu demandant l'action à effectuer (créer, détruire ou afficher) et un *conteneur* contenant les instances créées par l'utilisateur.

Le conteneur d'instances va être de type `Vector`, et va être un attribut de classe (mot-clé `static`). Cette classe est définie dans le package `java.util`.

## Afficher un menu, saisir le choix de l'utilisateur

Ecrire un programme qui affiche le menu ci-dessous, prend la commande de l'utilisateur au clavier, affiche un message « choix n » (n=1,2,3,0) et recommence tant que l'utilisateur ne veut pas quitter le programme.

1. créer un ballon
2. détruire un ballon
3. afficher la liste des ballons
0. quitter

## Créer une instance par l'intermédiaire du clavier et de l'écran

Enrichir le programme du TP précédent en remplaçant l'affichage « choix 1 » par l'appel à la méthode `creer_une_instance()` de la classe `Ballon`. Pour cela, on pourra avoir une interface de la classe `Ballon` du type suivant :

```
class Ballon {
    public String identificateur;           // identifier un ballon
    public int taille;                     // taille du ballon
    public Ballon(String s, int t) {}      // constructeur
    public static String lireIdentificateur() {} // dialogue
    public static int lireTaille() {}      // dialogue
    public String toString() {}           // l'instance en string
    public static Ballon creer_instance() {} // dialogue puis new
}
```

`String identificateur` est un attribut permettant d'identifier de manière unique une instance de la classe `Ballon`.

`taille` est un attribut descriptif d'une instance de la classe `Ballon`.

`lireIdentificateur()` est une méthode qui demande à l'utilisateur de rentrer un identificateur au clavier, qui le récupère et qui la retourne.

`lireTaille()` est une fonction qui demande à l'utilisateur de rentrer une taille au clavier, qui le récupère et qui le retourne.

L'objectif de ces deux dernières méthodes est très simple : retourner une valeur tapée au clavier. Pour cette raison, on peut choisir de ne pas les mettre sous forme de méthodes explicites. Elles doivent être exécutées *avant* d'appeler le constructeur. Ne pas confondre ces deux méthodes avec les méthodes d'accès, classiques en POO, `getIdentificateur()` et `getTaille()` d'accès aux attributs `identificateur` et `nom` lorsque ceux-ci sont privés.

`Ballon(String s, int t)` est le constructeur d'une instance de la classe `Ballon`. On suppose que l'on connaît l'identificateur et la taille du ballon avant de l'appeler. On passe en paramètres l'identificateur et la taille du ballon à créer.

`toString()` est la méthode de transformation d'une instance en string. Par exemple, pour une instance dont l'identificateur vaut « bienGonflé » et dont la taille vaut 4, `toString()` peut donner par exemple :

**identificateur = bienGonflé**  
**taille = 4**

`Ballon creer_une_instance()` est la fonction qui contient les appels à `lireIdentificateur()`, `lireTaille()`, `Ballon(String, int)` et `toString()`. Elle est appelée lorsque l'utilisateur tape 1 dans le menu `Ballon`. Elle fait exactement ce que faisait le programme principal du TP précédent.

## Afficher les instances à l'écran

Enrichir le programme précédent en remplaçant l'affichage « choix 3 » par l'affichage du vecteur des instances de la classe `Ballon`.

Pour cela, on ajoutera un attribut de classe `mesInstances` à la classe `Ballon`. On pourra utiliser les classes prédéfinies dans le paquetage `java.util` disponible en Java:

```
class Ballon {
    public static Vector mesInstances = new Vector() ;
    ...
}
```

Dans ce cas se reporter à la description de ces classes dans la documentation en ligne de Java.

Pour utiliser correctement `mesInstances`, on fera alors par exemple :

- ajouter une instance dans `mesInstances` dans le constructeur :

```
mesInstances.addElement(this);
```

- enlever une instance de `mesInstances` dans le destructeur :

```
mesInstances.removeElement(this);
```

## Détruire une instance par l'intermédiaire du clavier et de l'écran

Enrichir le programme précédent en remplaçant l'affichage « choix 2 » par l'appel à la fonction membre `détruire_une_instance()` de la classe `Ballon`. Pour cela, on pourra ajouter les membres suivants à la classe `Ballon` :

```
class Ballon {
    public void détruire() {} // 'destructeur'
    public static Ballon getInstance(String s) {} // retourne le ballon
    public static boolean détruire_instance() {} // dialog + détruire
}
```

`détruire()` est le 'destructeur' d'une instance de la classe `Ballon`. NB : les destructeurs ne sont pas nécessaires en Java. Cependant, pour plus de clarté, `détruire()` qui enlève la référence à l'instance correspond à un destructeur.

`Ballon getInstance(String s)` est la fonction qui retourne l'instance de la classe `Ballon` possédant un identificateur égal à la `String` passée en paramètre. Elle est appelée à chaque fois que l'on cherche une instance connaissant son identificateur. Elle retourne `null` si aucune instance ne correspond à la `String` passée en paramètre.

`boolean détruire_une_instance()` est la fonction qui contient les appels à `lireIdentificateur()`, `getInstance()`, `détruire()` et retourne `true` si la destruction est faite et `false` sinon. Elle est appelée lorsque l'utilisateur tape 2 dans le menu « `Ballon` ».

### Introduction

Ce TP reprend la suite du TP dans lequel l'utilisateur pouvait créer, détruire, afficher les instances d'une classe (la classe `Ballon`). Ici, nous allons permettre en plus à l'utilisateur d'*associer* et *dissocier* des instances appartenant à des classes différentes.

Pour cela, il nous faut plusieurs classes et un menu général permettant de diriger l'utilisateur vers la classe de son choix. Ce TP suppose qu'une classe `But` et que la classe `Joueur` ont été construites sur le principe de la classe `Ballon`. Pour commencer, il y a donc une partie importante d'édition et de couper-coller à faire dans ce TP. Ce TP suppose aussi que la classe `Vector` est connue (cf TP précédent).

Ensuite seulement, la partie intéressante du TP (associer, dissocier des instances) pourra être abordée.

### Plusieurs classes...

Ecrire un programme qui gère plusieurs classes. Par exemple les classes `Joueur`, `Ballon` et `But`. On pourra identifier un but par la minute à laquelle il a été marqué et on pourra décrire un but avec la distance à laquelle il a été marqué.

```
public int minute;  
public int distance;
```

Le programme affichera le menu ci-dessous :

1. **Joueur**
2. **Ballon**
3. **But**
0. **Quitter**

Puis, par exemple, si l'utilisateur a tapé 2, il affichera le menu :

1. **créer un ballon**
2. **détruire un ballon**
3. **afficher tous les ballons**
0. **précédent**

### Associer deux instances une à une

Enrichir le programme précédent en rajoutant la ligne suivante dans les menus `Ballon` et `But` :

4. **associer un ballon à un but.**

En particulier, on ajoutera un attribut et deux méthodes à la classe `Ballon` :

```
public But monBut;
public static void associer_but_instance() {} // dialog + associer
public void associer_but(But b) {} // assoc ballon but
```

Analogue pour la classe `But` :

```
public Ballon monBallon;
public static void associer_ballon_instance() {} //dial + assoc
public void associer_ballon(Ballon b) {} // assoc but ballon
```

### Associer une instance avec plusieurs instances

Enrichir le programme précédent en rajoutant la ligne suivante dans le menu `Ballon` :

#### **5. associer un ballon à des joueurs.**

Et la ligne suivante dans le menu `Joueur`:

#### **5 associer un joueur à un ballon.**

En particulier, on ajoutera un attribut a la classe `Ballon` :

```
public Vector mesJoueurs = new Vector();
```

Et un attribut à la classe `Joueur` :

```
public Ballon monBallon ;
```

Remarquer qu'il est nécessaire d'utiliser un attribut de type `vector` dans la classe `Ballon` pour associer des joueurs et un ballon car la spécification du projet STADE indique que l'ordre de multiplicité de l'association est « 0 ou plusieurs ». Par contre, cela n'est pas nécessaire dans l'autre sens.

### Dissocier des instances

Enrichir le programme précédent en rajoutant la ligne suivante dans le menu `Ballon` :

#### **6. dissocier un ballon d'un joueur.**

Et la ligne suivante dans le menu `Joueur`:

#### **6 dissocier un joueur d'un ballon.**

Compléter votre programme pour que tout lien puisse être dissocié.

## Imprimer des instances avec leurs liens

Pour visualiser les instances du programme, il est nécessaire d'avoir 2 méthodes qui transforment une instance en `String`. Une première méthode qui transforme une instance en une chaîne de caractères identifiant l'instance (nous l'appellerons `toIdent()`) et une seconde méthode qui transforme l'instance en une chaîne de caractères qui contient les noms des attributs de l'instance avec leur valeur, `toString()`.

Ecrire la méthode `toString()` et la méthode `toIdent()` pour toutes les classes de votre application. Par exemple, pour la classe `Ballon`, le corps de ces méthodes est :

```
public String toString() {
    if (monBut!=null)
        return "Identificateur " + identificateur + "\n" +
            "taille " + taille + "\n"
            + "But " + monBut.identificateur + "\n"
            + "Joueurs:\n" + mesJoueurs.toIdent();
    else
        return "Identificateur " + identificateur + "\n" +
            "taille " + taille + "\n"
            + "But " + "null" + "\n"
            + "Joueurs:\n" + mesJoueurs.toIdent();
}

public String toIdent() {
    return identificateur;
}
```

## Rendre le programme correct, robuste et plus facile à utiliser

A ce niveau du TP, vous avez une première version du programme qui permet de créer, détruire, associer, dissocier et afficher les instances, sans plus. Vous pouvez rendre robuste votre programme en vérifiant les tests suivants :

- créer et associer des instances, détruire l'une d'elle. afficher les instances non détruites. Que se passe-t-il ? Corriger le programme.
- associer une instance avec une instance déjà associée : le programme doit refuser l'association si elle est un-à-un.
- l'utilisateur veut détruire, associer ou dissocier une instance, le programme propose-t-il une liste des instances possibles ? Si la réponse est négative, améliorer le programme pour qu'il propose des listes d'instances à l'utilisateur lorsque cela est nécessaire.

### Introduction

Ce TP reprend la suite du TP dans lequel l'utilisateur pouvait créer, détruire, associer, dissocier des instances de différentes classes. A chaque session, l'utilisateur devait repartir de zéro. S'il avait créé des instances à la session précédente, il ne les retrouvait pas à la session suivante. Le but de ce TP est donc de *sauver* et *charger* sur *fichier* les informations contenues dans une classe. Ainsi, au début du nouvelle session, l'utilisateur pourra repartir des données de la fin de la session précédente. Ici, on suppose que la classe `Joueur` contient des joueurs. La classe contient une liste des joueurs de type `Vector`.

```
public class Joueur{
    public static Vector mesInstances = new Vector() ;
    public String nom;
    public int age;
}
```

### Ouvrir et écrire dans un fichier les instances d'une classe

Pour ouvrir un fichier (et pouvoir y écrire des informations de manière souple ensuite), on appelle successivement des constructeurs des classe `File`, `FileOutputStream` et `DataOutputStream`.

```
try{
    File fjoueurs = new File("Joueur.txt");
    FileOutputStream fosJoueurs=new FileOutputStream(fjoueurs);
    DataOutputStream dosJoueurs=new DataOutputStream(fosJoueurs);
    Joueur.sauverInstances(dosJoueurs);
}
catch(IOException e) {
    System.out.println("Problemes de Fichiers!");
}
```

Pour écrire effectivement dans le fichier, on utilise les méthodes `writeInt()`, `writeUTF()` de la classe `DataOutputStream`.

```
public static void sauverInstances(DataOutputStream dos) throws
IOException{
    dos.writeInt(mesInstances.size());
    for(int i=0;i<mesInstances.size();i++){
        Joueur j =(Joueur)mesInstances.elementAt(i);
        dos.writeUTF(j.nom);
        dos.writeInt(j.age);
    }
}
```

### Lire un fichier contenant les instances d'une classe

Pour ouvrir un fichier (et pouvoir y lire des informations de manière souple ensuite), on appelle successivement des constructeurs des classe `File`, `FileInputStream` et `DataInputStream`.

```
File sauvegardeJoueurs = new File("Joueur.txt");
FileInputStream fisJou = new FileInputStream(sauvegardeJoueurs);
DataInputStream disJou = new DataInputStream(fisJou);
Joueur.chargerInstances(disJou);
```

Pour lire des informations sauvegardées de la manière décrite ci-dessus dans un fichier, on utilise les méthodes `readInt()`, `readUTF()` de la classe `DataInputStream`.

```
public static void chargerInstances(DataInputStream dis) throws
IOException{
    int tailleFichier = dis.readInt();
    for(int i=0;i<tailleFichier;i++){
        String n = dis.readUTF();
        int a = dis.readInt();
        Joueur j = new Joueur(n, a);
    }
}
```

## **Intégrer les chargement et sauvegarde d'objets sur fichiers avec les TP précédents**

Le but de ce paragraphe est de mettre au point un programme qui :

- 1) lit des informations dans un fichier à propos d'une classe et crée les instances de cette classe en mémoire.
- 2) lit des informations tapées au clavier par l'utilisateur et crée d'autres instances de la classe en mémoire (cf. TP précédents).
- 3) écrit la liste des instances de la classe dans le fichier.

2 et 3 ont déjà été faits dans les TP précédents, il suffit donc de faire 1 et 3.

Rajouter la sauvegarde du fichier à la fin du programme (3). Mettre au point ce programme.

Rajouter le chargement du fichier au début du programme (1). Mettre au point ce programme en enrichissant le fichier à chaque exécution avec de nouvelles données.

Attention ! lorsque vous mettez au point ce programme, une fausse manipulation peut vous écraser le fichier `Joueur.txt`, donc sauvez `Joueur.txt` avant chaque nouvelle exécution... Voilà, vous savez désormais utiliser des fonctions qui permette de sauver des instances dans un fichier et de les charger depuis un fichier.

## **Ecrire/Lire des fichiers contenant des instances associées de classes différentes**

Lorsque les instances sont associées à d'autres le mécanisme précédent doit être amélioré avec deux passes. En effet lors de la lecture du fichier des instances, les instances associées à une instance ne sont pas nécessairement déjà construites. Il faut donc une première passe de lecture des fichiers en construisant les instances sans les associations, puis une deuxième passe sans la construction mais avec les associations.