

Bruno Bouzy

bouzy@math-info.univ-paris5.fr

27 mai 2003

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, MST1 et 2, IASV et licence d'informatique 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 :

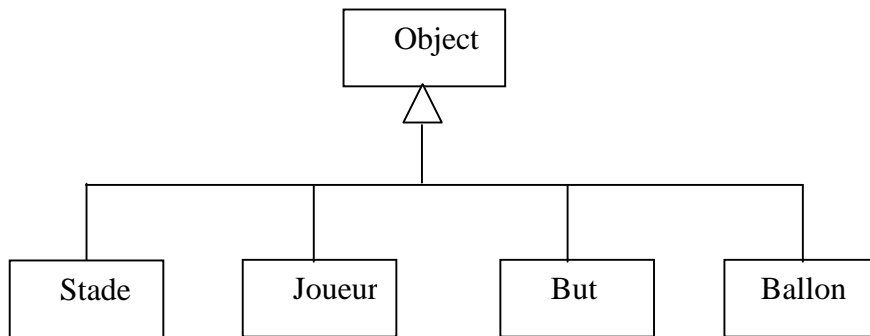
- un bref modèle UML du programme final `stade`,
- un TP "premiers pas",
- un TP "créer détruire des instances avec interaction utilisateur",
- un TP "associer dissocier des instances avec interaction utilisateur",
- un TP "charger sauver des instances sur fichier".

Introduction

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

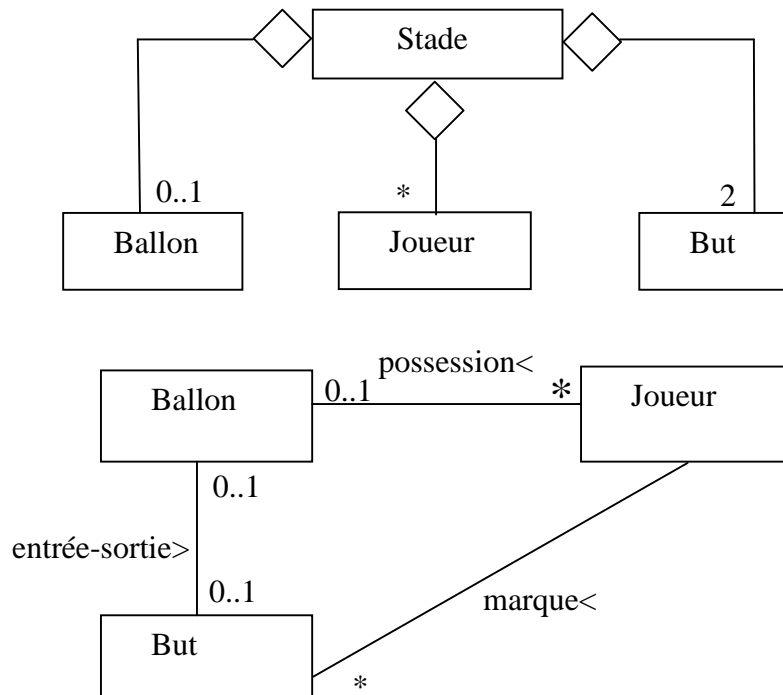
Rappels : les diagrammes globaux du logiciel STADE

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 ?

Si la fenêtre d'exécution disparaît immédiatement après son apparition, insérer le fichier `Keyboard.java` dans votre projet et rajouter la méthode `Keyboard.pause` à la fin du programme:

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

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 chaîne de caractères:");
        String s = Keyboard.getString();
        System.out.println("La chaîne 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 crée l'instance avec ses propriétés ; elle imprime l'instance sur la sortie standard. On suppose que la classe `Joueur` est définie comme suit :

Joueur
int age String nom
Joueur(String n, int a) toString() : String

`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`.

`toString()` est la redéfinition dans `Joueur` de la méthode `toString()` définie dans la classe `Object`.

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");
    System.out.println("Tapez un nom :");
    String s = Keyboard.getString();
    System.out.println("Tapez un age :");
    int a = Keyboard.getInt();
    Joueur j = new Joueur(s, a);
    System.out.println("Joueur j cree." + j);
    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.

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 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`.

`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 `Keyboard.getString()`, `Keyboard.getInt()`, `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 n'existent pas 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 `Keyboard.getString()`, `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 à 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..*". 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, on imprimera toujours les attributs intrinsèques (nom et age pour un `Joueur`). De plus, si un lien n'est pas nul (par exemple `if (monBut != null)`), on imprimera l'identificateur du lien (`monBut.minute`).

Pour afficher un `Ballon`, on pourra écrire et utiliser une méthode de classe `String Ballon.deMesJoueursAString()` retournant la `String` concaténation des noms des joueurs du vecteur `mesJoueurs`.

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 `Ballon` contient des ballons.

```
public class Ballon{
    public static Vector mesInstances = new Vector() ;
    public String identificateur;
    public int taille;
    public But monBut;
}
```

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 `FileWriter`, `BufferedWriter` et `PrintWriter`. Pour écrire effectivement dans le fichier, on utilise la méthode `println()`.

```
public static void sauveFichier() throws IOException {
    FileWriter fw = new FileWriter("Ballon.txt");
    BufferedWriter bw = new BufferedWriter(fw);
    PrintWriter pw = new PrintWriter(bw);
    for (int i = 0; i<mesInstances.size(); i++) {
        Ballon b = (Ballon) mesInstances.elementAt(i);
        pw.println(b.identificateur + " " + b.taille + " "
            + ((b.monBut!=null) ? String.valueOf(b.monBut.minute) : "-1")
        );
    }
    pw.close();
}
```

Lire un fichier contenant les instances d'une classe

On suppose que le fichier texte "Ballon.txt" contient les deux lignes suivantes :

```
bienGonfle 12 13
toutMou 1 26
```

(Un ballon d'identificateur "bienGonfle" de taille 12 lié au but marqué à la 13^{ème} minute et un ballon "toutMou" de taille 1 lié au but marqué à la 26^{ème} minute).

Pour ouvrir ce fichier (et pouvoir y lire des informations de manière souple ensuite), on appelle successivement le constructeur des classes `File` et `StreamTokenizer`. Pour lire le fichier, on utilise essentiellement la méthode `nextToken()` et l'attribut `nval` de la classe `StreamTokenizer`.

```

public static void chargeFichierInstance() throws IOException {
    FileReader fr;
    try { fr = new FileReader("Ballon.txt");
    } catch (FileNotFoundException e) { return; }
    StreamTokenizer st = new StreamTokenizer(fr);
    String s; int t; boolean encore = true;
    while (encore) {
        if (st.nextToken()==StreamTokenizer.TT_WORD) {
            s = st.sval;
            if (st.nextToken()==StreamTokenizer.TT_NUMBER) {
                t = (int) st.nval;
                Ballon b = new Ballon(s, t);
                if (st.nextToken()!=StreamTokenizer.TT_NUMBER)
                    encore = false;
            }
            else encore = false;
        }
        else encore = false;
    }
    fr.close();
}

```

Noter que cet exemple ne tient pas compte de la minute du but éventuel.

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

Mettre au point un programme qui charge `Ballon.txt` et crée les instances de la classe `Ballon`, dialogue avec l'utilisateur (cf. TP précédents), enfin écrit les instances de la classe `Ballon` dans le fichier `Ballon.txt`. Pour cela, rajouter la sauvegarde du fichier à la fin du programme:

```

// sauvegarde des instances
Ballon.sauveFichier();

```

Rajouter le chargement du fichier au début du programme :

```

// chargement des instances
Ballon.chargeFichierInstance();

```

Mettre au point ce programme en enrichissant le fichier `Ballon.txt` à chaque exécution avec de nouvelles données.

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

Lorsque les instances des classes `Ballon`, `But` et `Joueur` sont associées entre elles, il faut 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. Par exemple, le début du `main()` contient :

```

// chargement des instances
Ballon.chargeFichierInstance();
Joueur.chargeFichierInstance();
But.chargeFichierInstance();

// chargement des liens
But.chargeFichierLienJoueur();
Joueur.chargeFichierLienBallon();
Ballon.chargeFichierLienBut();

```

et la fin contient :

```
// sauvegarde des instances et des liens
Joueur.sauveFichier();
Ballon.sauveFichier();
But.sauveFichier();
```

Dans la classe `Ballon`, on peut définir le chargement des liens entre ballons et buts :

```
public static void chargeFichierLienBut() throws IOException {
    FileReader fr;
    try { fr = new FileReader("Ballon.txt"); }
    catch (FileNotFoundException e) { return; }
    StreamTokenizer st = new StreamTokenizer(fr);
    String s; int t; int b;
    boolean encore = true;
    while (encore) {
        if (st.nextToken()==StreamTokenizer.TT_WORD) {
            s = st.sval;
            if (st.nextToken()==StreamTokenizer.TT_NUMBER) {
                t = (int) st.nval;
                if (st.nextToken()==StreamTokenizer.TT_NUMBER) {
                    b = (int) st.nval;
                    if (b!=-1) {
                        But bu = But.getInstance(b);
                        Ballon ba = Ballon.getInstance(s);
                        bu.monBallon = ba;
                        ba.monBut = bu;
                    }
                }
            }
            else encore = false;
        }
        else encore = false;
    }
    else encore = false;
}
fr.close();
}
```

Noter que cette fonction suppose les instances déjà chargées. Elle est appelée *après* le chargement des instances.

Noter aussi que globalement les fichiers `Ballon.txt` `Joueur.txt` et `But.txt` contiennent tous les liens nécessaires au chargement, mais que individuellement il ne contiennent pas tous les liens. Par exemple, `Ballon.txt` contient les liens éventuels vers les buts et pas vers les joueurs. Les liens joueur-ballon sont contenus dans le fichier `Joueur.txt`.

En plus du fichier `ballon.txt` déjà donné avant, voici un exemple de fichiers texte sauvegardés :

```
But.txt :
13 13 zidane
26 26 henry
52 25 vieira
```

```
Joueur.txt:
zidane 30 bienGonfle
henry 25 toutMou
wiltord 28 bienGonfle
vieira 26 toutMou
```