EISTI se transforme et devient CY TECH	**INGE MAN 2A : EXO PREMIERE CLASSE BOUTEILLE SIMPLE**	GRENOBLE ECOLE DE MANAGEMENT

Rédigé par : Hervé de Milleville	Ref :*IM2A-EX-BOU-SIM*
A l'intention de : Ingé Man 2A	Créé le : 13/04/2020

1. Cahier de charges de la bouteille simple

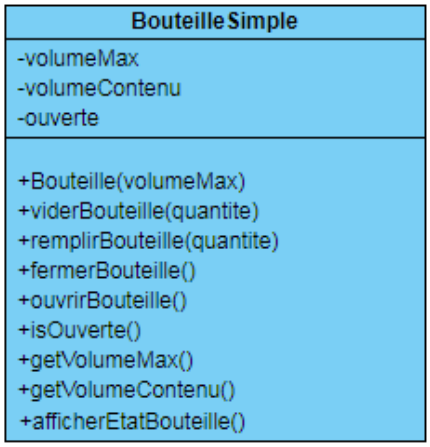
Une bouteille est définie par son volume maximum, son volume occupé en liquide et si elle est fermée ou ouverte.
On peut :

- ouvrir une bouteille,
- fermer une bouteille,
- remplir une bouteille à condition qu'elle soit ouverte et que cela ne dépasse pas sa capacité maximale,
- vider une bouteille à condition que la quantité à vider ne depasse la quantité contenue dans la bouteille,
- récupérer la valeur de chaque état de la bouteille,
- pouvoir afficher l'état de la bouteille.

Si on essaie de définir une « BouteilleSimple » avec un volume maximum négatif, on mettra par défaut ce volume à 1l.
A la fin de sa fabrication, une bouteille a son volume maximum donné par l'utilisateur et elle est vide et fermée.

2. Traduction UML

En fonction du cahier de charges précédent, le diagramme de la classe BouteilleSimple est le suivant :



Les symboles « - » et « + » signifient respectivement « privé » et « public ».
L'opération BouteilleSimple est un constructeur. Cett opération sera automatiquement appelée lors de la création d'un nouvel objet BouteilleSimple avec son volume maximum. Juste après cet appel, la nouvelle bouteille aura un volume maximum égal à volumeMax, sera vide et fermée.

3. Traduction Java

Vous pouvez récupérer les codes sources des différentes classes dans un fichier « .jar » sur la plateforme AREL.

3.1 La classe BouteilleSimple

```
public class BouteilleSimple {
  private float volumeMax;
  private float volumeContenu;
  private boolean ouverte;

  public BouteilleSimple(float volumeMax) {
    this.volumeMax = volumeMax;
    volumeContenu = 0.0f;
    ouverte = false;
  }
```

```
  public boolean viderBouteille(float quantite) {
    if(ouverte == false) {
      return false;
    }
    else if(quantite > volumeContenu) {
      return false;
    }
    else {
      volumeContenu -= quantite;
      return true;
    }
  }

  public boolean remplirBouteille(float quantite) {
    if(ouverte == false) {
      return false;
    }
    else if(quantite > (volumeMax - volumeContenu)) {
      return false;
    }
    else {
      volumeContenu += quantite;
      return true;
    }
  }

  public float getVolumeMax() {
    return volumeMax;
  }

  public float getVolumeContenu() {
    return volumeContenu;
  }

  public boolean isOuverte() {
    return ouverte;
  }
}
```

3.2 *Classe pour tester l'encapsulation des données de BouteilleSimple*

On écrit une classe qui essaie de violer volontairement l'encapsulation des données

```
public class TestEncasulationBouteilleSimple {
  public static void main(String[] args) {
    BouteilleSimple b;

    b = new BouteilleSimple(3);
    b.volumeMax = 5;
  }
}
```

Lors de l'exécution de cette classe, le message d'erreur suivant apparaît :

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  The field BouteilleSimple.volumeMax is not visible

  at TestBouteilleSimple.main(TestBouteilleSimple.java:7)
```

En revanche, si on met cette méthode main dans la classe BouteilleSimple et qu'on l'exécute, il n'y a aucune erreur. Cela est normal car le principe d'encapsultation s'applique quand on essaie d'utiliser directement un attribut d'une classe dans une opération (méthode) d'une autre classe. Dans le cas présent, la méthode main se trouve dans la classe BouteilleSimple.

3.3 Classe pour tester le bon fonctionnement de la classe BouteilleSimple

```
public class TestBouteilleSimple {
	public static void main(String[] args) {
		BouteilleSimple b; // On déclare une référence de BouteilleSimple

		b = new BouteilleSimple(3); // On alloue une instance de BouteilleSimple

		System.out.println("Etape 0");
		b.afficherEtatBouteille();

		System.out.println("\nEtape 1");
		if(b.remplirBouteille(2)) {
			System.out.println("On a pu remplir la bouteille");
			b.afficherEtatBouteille();
		}
		else {
			System.out.println("On n'a pas pu remplir la bouteille");
		}

		System.out.println("\nEtape 2");
		b.ouvrirBouteille();
		if(b.remplirBouteille(2)) {
			System.out.println("On a pu remplir la bouteille");
			b.afficherEtatBouteille();
		}
		else {
			System.out.println("On n'a pas pu remplir la bouteille");
		}

		System.out.println("\nEtape 3");
		if(b.viderBouteille(2.5f)) {
			System.out.println("On a pu vider la bouteille");
			b.afficherEtatBouteille();
		}
		else {
			System.out.println("On n'a pas pu vider la bouteille");
		}

		System.out.println("\nEtape 4");
		if(b.viderBouteille(1.5f)) {
			System.out.println("On a pu vider la bouteille");
			b.afficherEtatBouteille();
		}
		else {
			System.out.println("On n'a pas pu vider la bouteille");
		}

		System.out.println("\nEtape 5");
		b.fermerBouteille();
		b.afficherEtatBouteille();
	}
}
```

Après l'exécution de la méthode main de cette classe, l'affichage à l'écran donne :

```
Etape 0
Volume maximum de la bouteille 3.0
Volume contenu dans bouteille 0.0
La bouteille est fermée

Etape 1
On n'a pas pu remplir la bouteille

Etape 2
On a pu remplir la bouteille
Volume maximum de la bouteille 3.0
Volume contenu dans bouteille 2.0
La bouteille est ouverte

Etape 3
On n'a pas pu vider la bouteille

Etape 4
On a pu vider la bouteille
Volume maximum de la bouteille 3.0
Volume contenu dans bouteille 0.5
La bouteille est ouverte

Etape 5
Volume maximum de la bouteille 3.0
Volume contenu dans bouteille 0.5
La bouteille est fermée
```

4. Traduction C/C++ en procédural non objet

Vous pouvez récupérer les codes sources .cpp et .h dans un fichier « .zip » sur la plateforme AREL.

4.1 Le fichier BouteilleSimple.h

```
#include <iostream>
using namespace std;

struct BouteilleSimple {
  float volumeMax;
  float volumeContenu;
  bool ouverte;
};

void initialiserBouteilleSimple(struct BouteilleSimple & bs,float volumeMax);
bool viderBouteille(struct BouteilleSimple & bs,float quantite);
bool remplirBouteille(struct BouteilleSimple & bs, float quantite);
void ouvrirBouteille(struct BouteilleSimple & bs);
void fermerBouteille(struct BouteilleSimple & bs);
void afficherEtatBouteille(struct BouteilleSimple & bs);
float getVolumeMax(struct BouteilleSimple & bs);
float getVolumeContenu(struct BouteilleSimple & bs);
bool isOuverte(struct BouteilleSimple & bs);
```

4.2 Le fichier BouteilleSimple.cpp

```
#include "BouteilleSimple.h"

void initialiserBouteilleSimple(struct BouteilleSimple & bs,float volumeMax) {
  bs.volumeMax = volumeMax;
  bs.volumeContenu = (float) 0.0;
  bs.ouverte = false;
}

bool viderBouteille(struct BouteilleSimple & bs,float quantite) {
  if(bs.ouverte == false) {
    return false;
  }
  else if(quantite > bs.volumeContenu) {
    return false;
  }
  else {
    bs.volumeContenu -= quantite;
    return true;
  }
}

bool remplirBouteille(struct BouteilleSimple & bs, float quantite) {
                    if(bs.ouverte == false) {
                            return false;
                    }
                    else if(quantite > (bs.volumeMax - bs.volumeContenu)) {
                            return false;
                    }
                    else {
                            bs.volumeContenu += quantite;
                            return true;
                    }
}

void ouvrirBouteille(struct BouteilleSimple & bs) {
                    bs.ouverte = true;
}

void fermerBouteille(struct BouteilleSimple & bs) {
                    bs.ouverte = false;
}

void afficherEtatBouteille(struct BouteilleSimple & bs) {

                    cout << "Volume maximum de la bouteille " << bs.volumeMax << endl;
```

```
        cout << "Volume contenu dans bouteille " << bs.volumeContenu << endl;
        if(bs.ouverte) {
                cout << "La bouteille est ouverte" << endl;
        }
        else {
                cout << "La bouteille est fermée";
        }
}

float getVolumeMax(struct BouteilleSimple & bs) {
        return bs.volumeMax;
}

float getVolumeContenu(struct BouteilleSimple & bs) {
  return bs.volumeContenu;
}

bool isOuverte(struct BouteilleSimple & bs) {
  return bs.ouverte;
}
```

5. Approche procédurale versus approche objet

Les différences sont les suivantes :

- Structure et classe

Approche procédurale	Approche objet
On a défini une structure contenant les attributs de la bouteille simple. Par ailleurs on a défini les fonctions utiles (comportement) à la manipulation d'une bouteille simple.	la classe BouteilleSimple contient les attributs et les fonctions (ou méthodes).

- Fonctions et méthodes : prototypage

Approche procédurale	Approche objet
Toutes les fonctions ont comme premier paramètre une variable de type struct BouteilleSimple passé par référence et d'autres paramètres utiles à la fonction.	Toutes les méthodes ont un paramètre de moins que leur équivalent dans l'approche procédurale. Ce paramètre est la référence de l'objet concerné par le futur appel.

- Fonctions et méthodes : appel

Approche procédurale	Approche objet
On note bs la variable de type struct BouteilleSimple et f la fonction : *f(bs,liste des autres paramètres) ;*	On note bs la référence de la classe BouteilleSomple et f la méthode : *bs.f(liste des autres paramètres) ;*

INGE MAN 2A : EXO PREMIERE CLASSE BOUTEILLE SIMPLE

- Pour réserver et initialiser une variable ou un objet

Approche procédurale	Approche objet
On déclare une variable de type struct BouteilleSimple On appelle la fonction d'initialisation en passant en paramètres, la variable déclarée ci-dessus puis le volume maximum.	On déclare une référence de la classe BouteilleSimple Avec l'opérateur new appliqué à la référence déclarée, on alloue un objet en précisant son volume maximum (pour le constructeur appelé automatiquement).