

# BPO

## TP 2 - Iceberg2D




### Exercice 1 - Quelques schémas mémoire

<pre> package japan ;  public class Ysuka {     private char c ;     private Maya m ;     private Tsuki t ;      public Ysuka (char c, Maya m, Tsuki t) {         this.c = c ;         this.m = m ;         this.t = t ;     }      public Ysuka (Maya m) {         this.c = 'c' ;         this.m = m ;         this.t = new Tsuki(m.getVal()) ;     }      public Ysuka (Tsuki t) {         this.t = t ;         this.c = '.' ;         this.m = null ;     }      public Ysuka (Ysuka y) {         this.c = y.getC() ;         if (y.getM() != null) {             this.m = new Maya(y.getM()) ;         }         if (y.getT() != null) {             this.t = new Tsuki(y.getT()) ;         }     }      public char getC () {         return this.c ;     }      public Maya getM () {         return this.m ;     }      public Tsuki getT () {         return this.t ;     } } </pre>	<pre> package japan ;  public class Maya {     private int val ;      public Maya (int v) {         this.val = v ;     }      public Maya (Maya m) {         this.val = m.getVal() ;     }      public int getVal () {         return(this.val) ;     } }  package japan ;  public class Tsuki {     private int val ;     private Ysuka y ;      public Tsuki (int v) {         this.val = v ;         this.y = null ;     }      public Tsuki (int v, Ysuka y) {         this.val = v ;         this.y = y ;     }      public Tsuki (Tsuki t) {         this.val = t.getVal() ;         if (t.getY() != null) {             this.y = new Ysuka(t.getY()) ;         }     }      public int getVal () {         return this.val ;     }      public void setY (Ysuka y) {         this.y = y ;     }      public Ysuka getY() {         return this.y ;     } } </pre>
--	--

À l'aide du code des 3 classes **Ysuka**, **Tsuki** et **Maya** du package **japan** donné ci-dessus, dessinez à la main sur une feuille de papier, les schémas mémoire correspondant aux deux séquences de code suivantes :

code0	code1
<pre> Maya m1 = new Maya(5) ; Maya m2 = m1 ; Maya m3 = new Maya(m1) ; </pre>	<pre> Maya m1 = new Maya(5) ; Ysuka y1 = new Ysuka('y', m1, new Tsuki(10)) ; Tsuki t ; Maya m2 = new Maya(1) ; Ysuka y2 = new Ysuka(m2) ; </pre>

## Exercice 2 - Le logiciel artEoz

1. Depuis l'url **arteoz.loria.fr**, cliquez sur le lien “accès à la version en ligne du logiciel artEoz”.
2. À gauche, dans la fenêtre d'édition de code, copiez-collez **code1** donné à l'exercice précédent, à partir du fichier donné sur arche (le copier-coller à partir du fichier pdf de cet énoncé ne fonctionne pas avec les symboles).  
Le code entré correspond aux instructions de la fonction **main** d'une classe de test.
3. Cliquez sur le bouton  pour afficher le schéma mémoire correspondant à l'exécution de toutes les lignes du code donné.  
Le schéma affiché correspond-il à votre dessin fait à la main ?  
Cherchez les différences et essayez de comprendre. Aidez-vous des boutons qui permettent d'exécuter les instructions une par une (ce que l'on appelle le mode “pas à pas”) :   
4. On rappelle qu'un objet mort est une instance de classe qui n'est plus référencée par une variable ou un champ de classe. Ajoutez une ou plusieurs instructions dans le code afin de provoquer la présence d'un ou plusieurs objets morts et vérifiez leur présence avec l'option **Objets morts** sélectionné dans le menu ouvert par  ; cette option permet de visualiser en grisé les instances des classes qui ont été construites, mais qui ne sont plus accessibles.
5. Remplacez le code par celui-ci :

code2
<pre>Maya ma = new Maya(6) ;  for (int i = 0 ; i &lt; 4 ; i++) {     Maya m = new Maya(i) ;     int k = m.getVal() ; }</pre>

La variable **i**, indice de la boucle, est-elle encore accessible (utilisable) après l'itération ? Même question pour la variable **m**. Pourquoi ?

6. Combien y-a-t'il d'objets morts construits après l'exécution de ce code ? Vérifiez avec **artEoz**.
7. Remplacez le code par celui-ci :

code3
<pre>Maya m = new Maya(6) ; int k ; k = m.getVal() ;</pre>

Sélectionnez l'option **Pile** du menu des options  . Exécutez le code en mode “pas à pas”.  
Que pouvez-vous ainsi visualiser ?

8. À la main, dessinez le schéma mémoire correspondant au code suivant. Vous pouvez vérifier simultanément votre schéma avec celui produit par artEoz, en mode pas à pas.

code4
<pre>Maya m1 = new Maya(7) ; Maya m2 = m1 ; Maya m3 = new Maya(m1) ; Tsuki t1 = new Tsuki(0) ; Ysuka y1 = new Ysuka('f', m3, new Tsuki(45)) ; t1.setY(y1) ; Tsuki t2 = new Tsuki(t1) ;</pre>

### Exercice 3 - La classe Iceberg2D

1. Copiez chez vous, dans le bon répertoire, la classe **Iceberg2D** dont le code (incomplet) est donné sur Arche. Pour trouver le nom du répertoire dans lequel stocker le fichier, regardez le contenu du fichier... Quel est le nom complètement qualifié de cette classe ?
2. Cette classe utilise comme ressource l'archive **geometrie.jar**.

<b>Perso</b>	La ressource <b>geometrie.jar</b> est sur Arche. Recopiez ce fichier dans votre dossier <b>ressourcesBPO/geometrie</b>
<b>FST</b>	La ressource <b>geometrie.jar</b> est dans <b>/opt/depot/BPO</b> .  Surtout ne pas dupliquer cette ressource, il suffira de l'utiliser comme on a utilisé <b>awaz</b> au TP précédent.

3. La documentation de **geometrie.jar** est également fournie.

<b>Perso</b>	Placez la documentation de <b>geometrie</b> dans le dossier <b>ressourcesBPO/geometrie</b> .
<b>FST</b>	La documentation se trouve dans <b>/opt/depot/BPO/</b> .

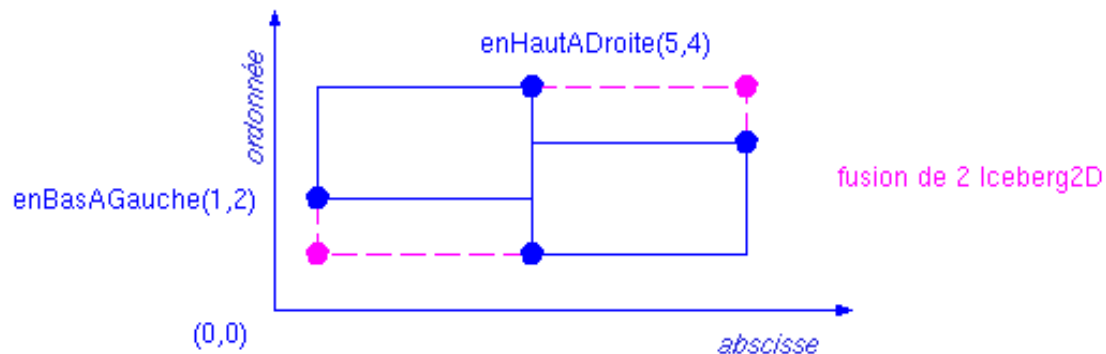
4. Générez la documentation de la nouvelle classe **Iceberg2D** avec la commande **javadoc** :

<b>Perso</b>	<b>javadoc -classpath ../ressourcesBPO/geometrie/geometrie.jar:. -d glaces/javadoc glaces</b>
<b>FST</b>	<b>javadoc -classpath /opt/depot/BPO/geometrie.jar:. -d glaces/javadoc glaces</b>


5. Créez la classe de tests **TestIceberg2D** dans le sous-package **tests**. Quel est le nom complètement qualifié de cette classe ? Ajoutez une fonction **main** dans cette nouvelle classe (attention au profil de cette fonction).
6. À l'aide de la documentation produite, complétez progressivement le code des fonctions de la classe **Iceberg2D** en parallèle avec la fonction **main** permettant de tester votre code. En pratique, vous commencez par écrire le code d'un constructeur et d'une première fonction d'observation. Puis vous écrivez une séquence de tests dans **TestIceberg2D**. Vous compilez l'ensemble et exécutez la classe de tests.

Si (et seulement si) cette étape est réussie, vous pouvez écrire et tester de la même façon une nouvelle fonction. Et ainsi de suite jusqu'à ce que toutes les fonctions de **Iceberg2D** soient écrites et testées.

- Chaque fonction est supposée travailler sur des paramètres corrects.
- Le schéma ci-dessous est destiné à vous aider dans la réalisation de la fonction qui permet de fusionner 2 icebergs.



7. Pour ceux qui disposent d'un compilateur Java 11, recompilez vos sources en Java11 et utilisez le logiciel **artEoz** pour tester votre application **Iceberg2D** :

- Chargez le fichier **Iceberg2D.class** à l'aide du bouton 
- Copier/coller une partie du corps de la fonction **main** de votre classe **TestIceberg2D** dans la fenêtre de code. Attention, il faut, avec **artEoz**, que les noms de variables commencent obligatoirement par une lettre minuscule.