L'objectif de ce stage est d'explorer les principales caractéristiques du langage Java

Remarques générales:

- *Installer un J2SE*
- *Installer un éditeur Java (www.jedit.org, …) assez puissant*
- *Installer ANT*
- *Utiliser un répertoire par séance*
- *Définir un ANT par exercice ou par séance exercice (pour compiler, lancer les tests, archiver et nettoyer)*
- *Utiliser au moins un fichier par exercice*

Documentation et Méthode de Travail

Pour vos séances de TDs, vous devez pouvoir consulter en ligne les APIs de Java. Celles ci sont dans le répertoire /usr/share/doclic/java

Vous pourrez également consulter :

- *Le tutoriel Java de Sun : /prof/ddonsez/pub/docs/java/tutorial*
- *Le livre Thinking in Java /prof/ddonsez/pub/docs/java/thinkingjava*
- *Les exemples des livres O'Reilly : /prof/ddonsez/pub/docs/exemplelivre*

S1 : MISE EN ROUTE

Concepteur : Didier DONSEZ
But : Mise en Route, Compilation, Package, JavaDoc, Manifest, Jar, tableaux, collections, String, StringBuffer.
Remarques :
Durée : 2H30

Ex1.1 : Mise en route et Compilation

Positionnez le CLASSPATH de votre machine si nécessaire (sous Unix et sous Windows32).
Ecrire un exemple simple d'application comportant 2 classes `Test` et `Queue`.
Compiler et Exécuter sous Unix. Quel est le résultat produit ?
Compiler et Exécuter sous Windows. Quel est le résultat produit ?

Ex1.2 : Premiers pas avec Java

La classe `QueueA` implante des files d'attente (de longueurs bornées) contenant des numéros (entiers)
Dans votre programme,
Ajouter les 2 méthodes put et get à la classe `QueueA`
Surcharger la méthode `toSrint` de `QueueA`
Définir un constructeur `QueueA(int len)` avec la longueur maximum de la file d'attente
Accès aux membres avec `this`
Créer plusieurs objets de cette classe. Affecter des références d'objets
Importer le package `java.io` pour les sorties écran. Comment se passer de l'import ?
Appel de méthodes
Implanter la file d'attente au moyen d'un tableau d'entiers
Que se passe t'il en cas de débordement du tableau (mauvais indice …) lors du parcours d'un tableau?
Manipuler les chaînes `String` et `StringBuffer` pour les affichages.

Ex1.3 : Documentation

Commenter votre code afin de générer la documentation JavaDoc de `QueueA`.
Générer la documentation JavaDoc de votre ensemble de classes et d'interfaces.
Parcourir la documentation générée avec votre navigateur.

Ex1.4 : Package

Créer un package `fr.votrenom.Queue` (mot clé package dans votre fichier)
Compiler. Quel est le résultat produit ?
Exécuter. Quel est le résultat produit ?

Ex1.5 : Archive JAR du package

Créer une archive JAR avec vos classes
Utiliser un fichier build.xml ANT pour sa construction
Compiler sous Unix. Quel est le résultat produit ?
Exécuter sous Unix. Quel est le résultat produit ?
Compiler sous Windows. Quel est le résultat produit ?
Exécuter sous Windows. Quel est le résultat produit ?
Ajouter y un manifeste pour en faire un exécutable
Exécuter depuis un autre répertoire. Quel est le résultat produit ?
Ajouter la documentation à votre archive JAR ainsi que vos sources

Ex1.6 : Conflits de nom de classe

Ecrire un exemple qui pose un conflit dans le nom de classes.
Compiler, Exécuter. Quel est le résultat produit ?
Manipuler le CLASSPATH.
Compiler, Exécuter. Quel est le résultat produit ?
Inverser les imports.

S2 : HERITAGE, CONSTRUCTEUR, PROTECTION

Concepteur : Didier DONSEZ
But : Héritage, Classe et Interface, Classe Imbriquée, Constructeur, Finalize, Clonage, Appel de méthodes par valeur , Surcharge, Surcharge Anonyme, Variables et Méthodes de classe, Classes finales et abstraites, Protection et Package.
Durée : 4H00

Ex2.0 : Documentation

Pour chaque exercice, penser à commenter votre code et a générer la documentation JavaDoc de votre ensemble de classes et d'interfaces. Parcourir la documentation générée avec votre navigateur.

Ex2.1 : Héritage simple

Définir un héritage simple de classes :

- une super-classe `Queue` de files d'attente d'objets de longueur bornée (dont les méthodes ne font rien mais contenant un membre `len`)
- une classe dérivée `QueueA` qui implante la file d'attente avec un tableau d'objets
- une classe dérivée `QueueV` qui implante la file d'attente avec un `java.util.Vector` d'objets

Ex2.1 : Héritage simple

Définir les constructeurs en n'oubliant pas d'utiliser `super`
Définir un constructeur sans argument avec un constructeur avec argument.
Créer des objets de ces 3 classes mettant en attente des numéros (entiers). . Affecter les à des références du type de la super-classe.

Ex2.2 : Classes finales et abstraites

Rendre la super-classe abstraite `Queue`.
Créer un objet de la super-classe. Compiler. Que se passe t'il ?
Rendre les méthodes `get` et `put` abstraites. Que se passe t'il si une classe dérivée ne redéfinit pas ces méthodes ?
Rendre finale la classe dérivée `QueueA` et la classe `Queue`
Dériver une classe de cette classe. Compiler. Que se passe t'il ?
Rendre finale la classe dérivée `QueueA` seule.
Dériver une classe de cette classe. Compiler. Que se passe t'il ?

Ex2.3 : Interface

Nous allons proposer une alternative de conception à l'héritage du 2.1 en utilisant les interfaces.
Définir une interface et deux classes pour remplacer l'héritage 2.1.
Placer des traces. Créer des objets. Affecter les à des références du type de l'interface.
Compiler et exécuter
Modifier ces classes pour qu'il soit possible d'utiliser `java.util.Enumeration` pour parcourir les éléments des files d'attentes. Remarque : c'est très simple pour `QueueV`; .il faut se creuser les méninges pour `QueueA` (Indication : utiliser une classe imbriquée qui "implémente" `java.util.Enumeration`

Ex2.4 : Surcharge et Surchage Anonyme

Ecrire une classe de base et une classe héritée avec des méthodes surchargés.
Compiler, Exécuter. Quel est le résultat produit ?
Créer une objet avec une surcharge anonyme de méthode (à partir du JDK1.2).
Compiler, Exécuter. Quel est le résultat produit ?

Ex2.5 : Membres et Méthodes de classe

Ajoutez un membre de classe numInstances qui compte le nombre d'instance de Queue. Initialisez le membre de classe à zéro
Ajouter une méthode de classe incrNumInstance() qui incrémente la variable numInstances
Ajouter une méthode de classe getNumInstance() qui consulte la variable numInstances
Ajoutez un membre de classe numInstances qui compte le nombre d'instance de QueueA. Initialisez le membre de classe à zéro
Ajouter une méthode de classe incrNumInstance() qui incrémente la

variable numInstances
Ajouter une méthode de classe getNumInstance() qui consulte la variable numInstances.

Ecrire un fichier de test qui teste ces méthodes.

Ex2.6 : Protections

Définir un package.
Positionner les modifiers de manière à rendre privée la variable de classe, accessible
Tenter d'accéder à ces membres et à ces méthodes depuis une classe dérivée, depuis une classe du package, depuis une classe externe au package.

Ex2.7 : Manipulation des erreurs avec les exceptions

Redéfinir l'interface Queue et ses implémentations pour que la méthode get lève une exception quand la file est vide et pour que la méthode put lève une exception quand la file est pleine.
Définir un héritage d'exception `QueueException`, `QueueIsFullException`, `QueueIsEmptyException`.
Tester avec un programme d'exemple.

Ex2.8 : Un cas extrême avec les exceptions

Il est possible d'effectuer une sortie de boucle sans test d'arrêt mais en levant une exception. Ecrire ce cas et comparer le temps pris avec une solution classique.
Faites la même chose avec un parcours de tableau.

Ex2.9 : Finalize & Garbage Collector

Définir un finalize pour ces classes. Ne pas oublier `super.finalize()`
Quand celui-ci se déclenche ?
Faire une bouche de création de plusieurs milliers d'objets. Arrêter la boucle au 50ème finalize.
Déclenchez le directement.
Qu'est ce qui change quand Runtime.RunFinalizationOnExit(true) est exécuté?
Forcer le garbage collector.

Ex2.10 : Clonage

Ecrire une classe clonable ObjClonable et une classe non clonable ObjNonClonable dérivant du classe Obj avec un membre String et une méthode `toPrint()`
Rendre la classe `QueueA` clonable
Créer des objets de la classe clonable et un objet de la classe non clonable
Ajouter ces objets à un objet `QueueA`
Cloner cet objet `QueueA`. Afficher les contenu de l'objet QueueA et de son clone.
Modifier un des objets contenus dans le clone. Afficher les contenu de l'objet `QueueA` et de son clone.
Que se passe t'il ?
Créer un objet de la classe non clonable. Ajouter cet objet à un objet `QueueA`
Cloner cet objet `QueueA`.
Recommencer l'exercice avec `QueueV`.

Comment faire un appel de méthode par valeur ?
Compiler, Exécuter

Ex2.11 : Liste d'objets de la même classe

Java ne supporte pas les types génériques (template) comme en C++ ou Ada.
Proposer deux solutions pour avoir dans une liste des objets de la même classe.
Même question avec les objets de classe dérivées.

Ex2.12 : classes Class et Method

Inspecter (i.e. afficher) la classe d'un objet et ses méthodes au moyen des classes Class et Method

Ex2.13 :Diffusion de l'initialisation

L'initialisation d'un objet peut être définie en dehors du constructeur défini dans la classe. Ecrire un exemple et tester

Ex2.14 : Documentation

Commenter votre code afin de générer la documentation JavaDoc
Générer
la documentation JavaDoc de votre ensemble de classes et d'interfaces.
Parcourir la documentation générée avec votre navigateur.

S3 : I/O, SERIALISATION

Concepteur : Didier DONSEZ
But : Exceptions, I/O, Sérialisation
Documents : Livre " Java I/O ", Tutorial en ligne
Durée : 2H00

Ex3.1 : Lecture formatée au clavier

Lire un entier, un flottant, une chaîne et une date au format DD/MM/YYYY au clavier puis les sortir sur la sortie standard System.out.

Ex3.2 : Ecriture d'un Journal

Ecrire un utilitaire d'écriture `writelog` en fin de fichier d'une suite d'enregistrement comportant la date courante, l'identificateur de l'écrivain et de son commentaire. L'identificateur de l'écrivain est saisit avant la saisie du premier commentaire. Le journal est partagée entre plusieurs écrivains (vous le testerez en utilisant plusieurs fenêtres). Attention aux fermetures de fichier !

Conseil : Définir une classe Entry ayant pour membres la date courante, l'identificateur de l'écrivain et de son commentaire et une méthode d'écriture dans une OutputStream ouvert.

Quels problèmes peut poser la concurrence d'accès par plusieurs écrivains (writelog) ? Proposer des solutions pour éviter ces problèmes.

Ex3.3 : Lecture d'un Journal

Ecrire un utilitaire `scanlog` de parcours sélectif du journal. Les critères optionnels de sélection sont :

`-after` D :
pour afficher les enregistrements écrits après la date D
`-before` D :
pour afficher les enregistrements écrits avant la date D
`-user` I :
pour afficher les enregistrements écrits par l'écrivain identifié par l'identificateur I.

Les options peuvent se combiner entre elles

```
scanlog -user donsez -user lecomte ../access.log
scanlog -after 01/12/1999 \
     -before 01/02/2000 ../access.log
cat ../access.log | scanlog -after 01/12/1999
```

Conseil : Compléter la classe Entry d'une méthode de lecture dans une InputStream ouvert.

Quels problèmes peut poser la concurrence d'accès par plusieurs écrivains (writelog) et lecteurs (scanlog) ? Proposer des solutions pour éviter ces problèmes.

Ex3.4 : Sérialisation du Journal

Ecrire les classes Log et Entry qui représente un journal. La classe Log est un vector d'Entry.
Ecrire les classes sérialisables LogS et EntryS dérivant des classes Log et Entry.
Réécrire les utilitaires writelog et scanlog en utilisant ces classes.
Quels problèmes peut poser la concurrence d'accès par plusieurs écrivains (writelog) et lecteurs (scanlog) ?

Ex3.5 : Externalisation du Journal

Définir des classes externalisables LogE et EntryE pour Log et pour Entry.
Réécrire les utilitaires writelog et scanlog en utilisant ces classes.

Ex3.6 : Mélange d'externalisation et de serialisation du Journal

Définir une classe externalisable LogES pour Log. Cette classe utilise la classe serialisable EntryS d'Entry. (inspirez vous de l'exemple 11.5 donné dans le livre " Java IO ").
Réécrire les utilitaires writelog et scanlog en utilisant ces classes.

Ex3.7 : Journal Chiffré (optionnel).

Réécrire les utilitaires writelog et scanlog pour manipuler un fichier chiffré. La clé génératrice est passée argument avec l'option -deskey. (inspirez vous de l'exemple 11.7 donné dans le livre " Java IO ").

Ex3.8 : Fichier de configuration

Charger un fichier de propriétés et afficher les paires atrtribut-valeur

S4 : DEBUGGER, THREADS

Concepteur : Didier DONSEZ
But : Debugger, Threads
Documents : Livre " Java Thread ", Tutorial en ligne
Durée : 2H30

Ex4.1 : Utilisation du debugger

Ecrire un programme qui boucle dans un appel de méthode
Utiliser le débugger pour consulter l'état du programme et pour repérer l'erreur
Faites la même chose avec un parcours débordant du tableau.

Ex4.2 : Thread

Ecrire une classe `Countdown` dérivant de la classe `Thread` et qui boucle N fois en affichant le numéro du tour après une pause P. N et P sont choisies à l'initialisation de la thread.
Ecrire un programme qui instancie trois threads `Countdown`.
Compiler, Exécuter

Ex4.3 : Thread Deamon

Reprendre le programme précédent mais transformer une des threads (la plus lente) en démon (daemon).
Que s'est il passé par rapport au 4.2

Ex4.4 : Héritage

On veut maintenant pouvoir utiliser plusieurs comportements de décompte. Ces comportements diffèrent dans la durée du sommeil. Créer deux classes dérivées `FixedSleepCountdown`, `RandomSleepCountdown` d'une super-classe abstraite `CountDown` qui ont une durée de sommeil fixé à l'initialisation de la thread ou un sommeil tiré aléatoirement avant chaque sommeil. La méthode `run()` sera définie et finale dans la super-classe

Ex4.5 : Interface Runnable

Utiliser l'interface `Runnable` pour utiliser des threads dans la création de classes dérivées `RandomSleepCountdown`, `FixedSleepCountdown` d'une super-classe abstraite `CountDown` qui ne dérive pas de la calsse `Thread`.

Ex4.6 : Synchronisation des threads : Rendez Vous Producteur/Consommateur

Créer une super-classe `Worker` et une classe `RendezVous` pour simuler une chaîne de production à plusieurs étages
Par exemple : N étapes S1=2 producteurs, S2=3 intermédiaires, S3=4 consommations.
Paramêtrer les temps de traitements de chaque étage (loi et paramêtres)

Ex4.7 : Synchronisation des threads : Files d'attente

Même exercice que le précédent mais avec une file d'attente au lieu d'un RendezVous entre chaque étage de travailleur
Définir la classe file d'attente `QueueS` utilisable par plusieurs threads concurrentes (utiliser `Wait` et `Notify`). Indication : `QueueS` uitlise `QueueA`.
Quelles sont les alternatives de conception ?

Ex4.8 :Interblocage : Le Diner des Philosophes

Modéliser et implanter le diner des philosophes.
Utiliser la classe `java.Math.Random` pour choisir aléatoirement la fourchette droite ou gauche en premier.
Tracer les méthodes.
Que se passe t'il au bout d'un certain temps d'exécution ?

S5 : APPLET ET AWT

Concepteur : Didier DONSEZ
But : Manipulation des Applets et d'AWT Simple
Documents : Livre " Java AWT ", Tutorial en ligne
Durée : 2H30

Ex5.1 : Une Applet Simple

Ecrire une applet simple qui affiche "Bonjour, je suis une applet"
Insérer la dans une page HTML

Ex5.2 : Paramêtre d'une Applet

Ecrire une applet affichant les paramêtres qui lui sont passés dans un TextArea.
Insérer la dans une page HTML en 2 endroits avec des paramêtres différents.

Ex5.3 : Applet et JAR

Archiver les classes de votre applet dans une archive
Réécriver la page HTML qui contient votre applet en utilisant l'archive créée

Ex5.4 : Cycle de vie d'une Applet

Ecrire une applet qui trace les changements d'état dans sa vie. La trace est affichée dans un TextArea et sur la sortie standard System.out.
Insérer la dans une page HTML contenant des liens vers d'autres documents. Que se passe t'il lorsque l'utilisateur (vous) naviguez vers ces liens puis lorsque vous revenez en arrière (back) ?

Ex5.5 : Applet et Sécurité

Ecrire une applet qui tente soit de lire un fichier local soit d'écrire dans un fichier local (`/etc/passwd` par exemple). L'action (READ ou WRITE) et le nom du fichier sont passés en paramêtre.
Que se passe t'il ? Pourquoi ?

Ex5.6 : Un éditeur simple avec les AWT

Programmer un éditeur de dessin (simple) qui permette de dessiner plusieurs formes de 4 types : texte, ellipse, rectangle, ligne.

Faire en sorte que cet éditeur puisse s'exécuter comme une applet Java ou comme une application Java.

Utiliser l'héritage pour définir la hiérarchie de classes suivante

- MyShape : superclasse abstraite (membre couleur de fond et couleur de trait)
- MyText : une ligne de texte
- MyRectangle : un rectangle
- MyEllispe : une ellipse
- MyNamedRectangle : un rectangle avec un texte centré
- MyNamedEllispe : : une ellipse avec un texte centré

Les méthodes de MyShape sont:

- boolean contains(int x,int y) qui teste si les coordonnées sont dans la forme (utilisé pour la sélection des formes).
- void paint(java.awt.Graphics g) qui peint la forme en fonction de son état courant (SELECTED, NOTSELECTED).
- void move(int deltax,int deltay) qui déplace la forme d'une position relative (deltax, deltay)
- d'autres méthodes peuvent être nécessaires.

Les formes sont ordonnés par plan (i.e. une forme en avant plan peut cacher une forme en arrière plan.

Ajouter un bouton qui génère 3 nouvelles formes (aléatoirement) de trois couleurs différentes.

Permettre le déplacement des formes au moyen de la souris (la dernière forme saisie est en avant plan). Utiliser la méthode contains pour détecter la forme saisie et en surchargeant les méthodes mouseDown, mouseMove, mouseUp de l'applet.

Remarque : Pour cet exercice, vous pouvez regarder l'applet `GraphLayout` fournie dans les démonstrations du JDK.

S6 : SWING

Concepteur : Didier DONSEZ
Documents : Livre " Java AWT ", Livre " Java Swing ", Tutorial en ligne
Durée : 4H00

Ex6.0 :Exploration de Swing

Explorer la démonstration Swing SwingApplet

Ex6.1 : Un éditeur de dessin avec Swing

Utiliser Swing pour enrichir l'éditeur de dessin simple développé avec les AWT. Reprendre les spécifications et la conception orientée objet de l'éditeur de dessin et y ajouter une barre de menu et une barre d'outils.

La barre de menu contient :

un menu "File" avec
- "New" qui crée un nouveau dessin vierge.
- "Open …" qui permet d'ouvrir un dessin sauvegardé dans un fichier.
- "Save" qui sauvegarde le dessin
- "Save as …" qui sauvegarde le dessin sous un autre nom.
- "Export SVG …" qui sauvegarde une représentation SVG du dessin.
- "Quit" qui termine l'application après avoir vérifier si des modifications n'avaient pas été sauvegardés.

un menu "Edit" avec
- "Cut" qui retire les formes sélectionnées du dessin et les place dans le presse-papier.
- "Paste" qui clone les formes sélectionnées du dessin et place ces clones dans le presse-papier.
- "Copy" qui clone les formes du presse-papier et place ces clones dans le dessin.

un menu "Tools" qui correspond aux outils de la barre d'outils
un menu "Help" qui fait apparaître une boite de dialogue.

La barre d'outils contient
- un outil "Fleche" pour sélectionner ou desélectionner une ou plusieurs formes.
- un outil "Rectangle" pour créer un nouveau rectangle.
- un outil "Ellipse" pour créer une nouvelle ellipse.
- un outil "Line" pour créer un nouveau ligne.
- un outil "Text" pour ajouter un texte.
- un outil "Couleur de Trait" pour changer la couleur de fond des formes sélectionnées et des nouvelles formes créées.
- un outil "Couleur de Fond" pour changer la couleur de fond des formes sélectionnées et des nouvelles formes créées.

Le dessin est sauvé dans un fichier en sérialisant les formes contenus dans le dessin. L'ouverture d'un dessin utilise la desérialisation.

Les outils de changement de la couleur utilisent une boite de dialogue commune.

Le format SVG pour l'exportation du dessin est défini au www.w3.org. Le format PGML pour l'exportation du dessin est défini au www.w3.org. En voici un exemple:

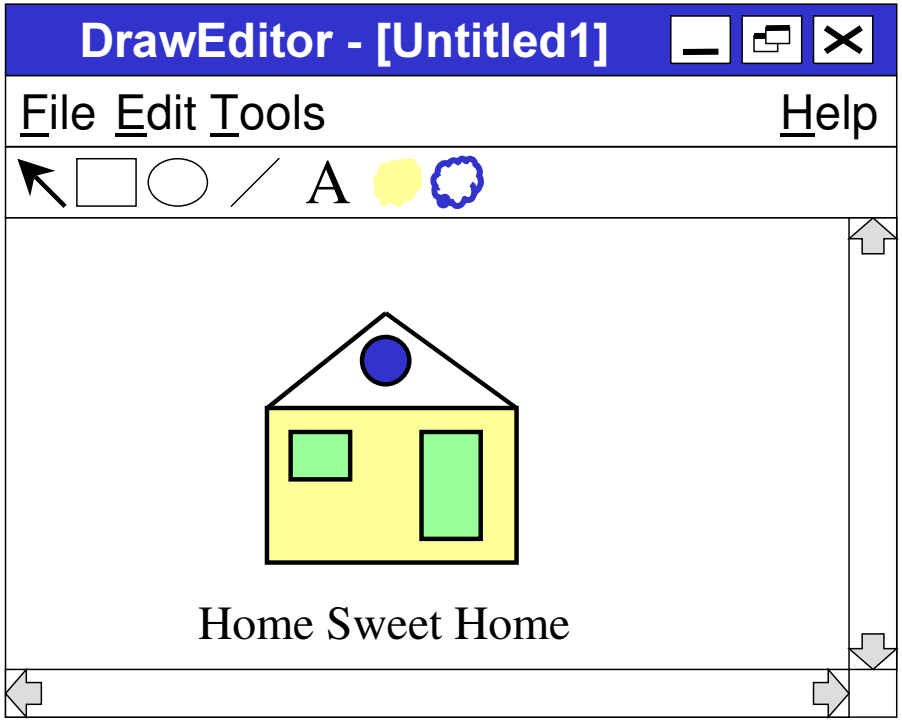
```
<?xml version="1.0"?>
<!DOCTYPE pgml SYSTEM "pgml.dtd">
<pgml description="fr.univ-valenciennes.desstnsi.DrawEditor"
      name="schemaDemo.ser"
>
    <rectangle name="Fig1"
      x="27" y="40" width="54" height="85"
      fill="1" fillcolor="-8355712"
      stroke="1" strokecolor="-8355712"
    />
    <ellipse name="Fig2"
      x="54" y="60" rx="18" ry="15"
      fill="1" fillcolor="-1"
      stroke="1" strokecolor="-16777216"
```

```
    />
</pgml>
```

Ex6.2 : Un éditeur de dessin multidocument

Ajouter à l'éditeur précédent la possibilité des manipuler plusieurs documents simultanément. Utiliser les internals frames.

Annexe : Interface de l'éditeur



S8 : BEANS, BDK ET BEANBOX, VISUALAGE, JBUILDER

Concepteur : Didier DONSEZ
But : Création de composants JavaBean
Durée : 4H00

Ex8.1 : Tutorial BeanBox

Suivre le tutorial BeanBox de Sun pour assembler des Beans

Ex8.2 : Création de Composant Bean

Ex8.2.1 : Potentiomêtre

Ecrire un composant JavaBean de potentiomêtre rond (le potentiomêtre droit existe déjà dans Swing sous la forme d'un slider).

Ex8.2.2 : Indicateurs de Contrôle : Compteurs et Jauges

Ecrire un composant JavaBean pour une des indicateurs suivants (le choix se fera par tirage au sort !) :

- Boussole, Indicateur d'assiette (roulis, tangage)
- Tachymètre (Vitesse), Compte-tour, Température d'huile, Température d'eau, …
- Jauges continus et discrets (multicolore, clignotement si surcharge, …)
- Chronogramme (utilisable pour historiser des performances, des ECG, …)

Ces indicateurs peuvent être rendus génériques en créant des super-classes Aiguille, Graduation, Cadran, … que vous dérivez pour obtenir des formes différentes.

Ex8.3 : Ajout de Composant Bean à BeanBox

Suivre le tutorial BeanBox de Sun pour assembler des Beans que vous avez créés

Ex8.4 : Composants Bean multithreadés

Utilisez le multithreading pour rendre clignotant (changement alternatif de couleurs) les indicateurs de contrôle en cas de surcharge.

Pour le chronogramme, afficher régulièrement la moyenne (ou le maximum, le minimum, la dernière valeur ou la première valeur, ca doit être paramêtrable) des dernières valeurs entrées depuis le dernier rafraîchissement.

Utilisez le multithreading pour rendre "trainante" les aiguilles des indicateurs de contrôle (le multitreading est obligatoire pour le Chronogramme).

Annexe : Idées d'aiguilles et d'indicateurs.