ESISAR3

TP de Programmation Orientée Objet n°1

Introduction aux outils du SDK
&
Aux constructions algorithmes de base

Dernière modification : juillet 2010

(*Philippe.Morat@imag.fr*)

Environnement nécessaire à la programmation java

Java development kit installé (notamment la commande `javac`) ; interprète java (`java`) ; un éditeur quelconque (nedit, gedit, …).

pour en savoir plus sur : http://www.oracle.com/technetwork/java/

Un programme Java peut prendre deux formes distinctes, chacune étant adaptée à un contexte d'invocation et d'exécution différents.

- La première (applications java) permet de créer des applications au sens classique du terme, c'est à dire des programmes s'exécutant de manière autonome à l'aide d'un interpréteur Java.
- La deuxième (applets Java) est destinée à des programmes invoqués depuis des documents HTML (HyperText Markup Language) diffusés sur le World Wide Web (WWW) et exécutés à l'intérieur d'un navigateur (Mozilla, Microsoft Internet Explorer, …) ou d'un « visualiseur » d'applets équipé d'un interpréteur Java (par exemple l'application `appletviewer` du SDK).

Les exercices qui vous sont proposés ici ont pour but de vous faire expérimenter ces deux types de programmes Java et de vous faire utiliser les outils de base du SDK ainsi que de vous familiariser avec :

- le compilateur (*`javac`*),
- l'interpréteur Java (*`java`*) pour les applications "stand-alone",
- *`appletviewer`* pour l'exécution d'applets
- *`jar`* pour créer des archives de fichiers de byte-code et de ressources
- *`javadoc`* pour générer de la documentation html à partir des commentaires documentant se trouvant dans le code source des classes.
- Les constructions langagières permettant de réaliser des algorithmes fondamentaux (affectation, alternative, itérative, …).

Exercice 0 : consulter la documentation en ligne pour le cours AOO

Avant de commencer le TP survoler cette documentation en quelques minutes, **savoir chercher dans celle-ci vous sera indispensable pour la suite**.
A titre d'exercice vous pouvez essayer de retrouver :

- la documentation de la classe `Math` de l'API java, classe définie dans le package `java.lang`
- la documentation de la classe `Vector`
- la documentation du compilateur `javac`
- le tutorial sur les fonctionnalités (features) du langage concernant les structures de données (Collections)

Exercice 1 : compiler et exécuter une application java

pour en savoir plus sur : le compilateur `javac` et l'interpréteur `java`

Comme dans tout système, l'environnement Java fixe le point d'entrée (début du programme) de façon implicite. C'est le rôle de la méthode main dont la signature est imposée : `public static void main(String[] args)`. Toute classe possédant une telle méthode peut être exécutée par l'interprète java.

1. récupérez le programme source `Premice.java`,
2. compilez et exécutez l'application `Premice.java`,
3. modifiez ce programme de manière à ce qu'il provoque l'affichage suivant :
   ```
   BONJOUR
   CECI EST MON
   PREMIER PROGRAMME JAVA
   ```
4. modifiez ce programme de manière à ce qu'il imprime `JE SOUSSIGNE, xx, …` où xx est votre nom que vous passerez comme premier argument de la commande.
5. Faites une seconde exécution où xxx sera de la forme `NOM PRENOM`.
6. modifiez ce programme de manière à ce qu'il imprime `JE SOUSSIGNE, xx né(e) le jj/mm/aa,…` où xx, jj, mm et aa est sont les arguments de la commande.

Indiquez sur une échelle de 4 la difficulté que vous avez rencontrée à faire cet exercice :
(1 : très facile, 2 : facile, 3 : difficile, 4 : trop difficile)

Exercice 2 : Mise en œuvre d'une application simple

pour en savoir plus sur :

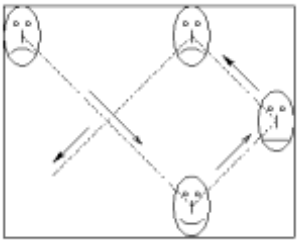
le générateur de documentation `javadoc`
le class path : pour définir les règles de recherche des byte-codes des classes java.
la construction de fichiers jar (fichiers Java ARchives)

1. Créer un répertoire `Visages` avec deux sous répertoires `src` et `classes`
2. Sauvegardez dans le répertoire `src` les fichiers suivants : `VisageRond.java`, `Dessin.java` et `AppliVisage1.java`
3. Placez vous dans le répertoire `src` et compilez l'application `AppliVisage1` en utilisant la commande `javac -d ../classes AppliVisage1.java` pour placer les fichiers de byte-code dans le répertoire `classes`.
4. Placez-vous dans le répertoire `classes` et constatez que l'ensemble des classes nécessaires a été compilé.
5. Depuis ce répertoire classes exécutez l'application `AppliVisage1`. Faire CTRL C sur la ligne de commandes pour arrêter l'application.

6. Replacez-vous dans le répertoire `Visages` et exécutez l'application `AppliVisage1` en utilisant l'option `-classpath (ou -cp)` pour indiquer à l'interpréteur `java` où trouver les classes à charger.
7. Déplacer le fichier `AppliVisage1.class` du répertoire classes vers le répertoire `Visages` (commande `move` sous MS-DOS ou mv sous unix).
8. Essayez de réexecuter l'application `AppliVisage1` par la commande `java AppliVisage1`, constatez.
9. Replacez-vous dans le répertoire `classes` et construisez un fichier `Visages.jar` contenant les fichiers byte-code des classes `s'y trouvant`, fichier jar que vous placerez dans le répertoire `visages`.
10. Replacez-vous dans le répertoire `Visages` et lancez à nouveau l'exécution en utilisant ce fichier jar dans l'option `-classpath` de la commande `java`.
11. Pour comprendre le fonctionnement de cette application étudiez la documentation HTML des classes `VisageRond` et `Dessin` que vous générerez à partir des commentaires documentants situés dans les sources de ces classes (regardez ces sources).
    a. créez un répertoire `docs` dans le répertoire `Visages`
    b. depuis le répertoire `src` lancez la commande `javadoc -d ../docs VisageRond.java Dessin.java`. Les fichiers html de documentation sont générés dans le repertoire `docs`, le point d'entrée étant le fichier `index.html`.
    c. consultez cette documentation à l'aide d'un navigateur. La documentation de la classe VisageRond référence une image que vous trouverez ici : `figurevisage.gif`. Pour que la documentation soit complète recopiez cette image dans le répertoire `docs`.
12. Modifiez la classe `VisageRond` afin que la taille par défaut d'un visage soit de 100x100 au lieu de 50x50 et recompilez cette classe.
13. Réexecutez l'application `AppliVisage1` en utilisant la même commande qu'au point 10, constatez.
14. Réexecutez l'application en application la commande du point 4, constatez.
15. Sauvegardez dans le répertoire src le fichier : `AppliVisage2.java`
16. Placez vous dans le répertoire `src` et compilez AppliVisage2 avec la commande `javac -classpath ../classes -d .. AppliVisage2.java` , corrigez les erreurs de compilation.
17. Exécuter l'application AppliVisage2, que constatez-vous ?
18. Corrigez et exécutez l'application `AppliVisage2`.
19. Consulter la documentation de la class Dialogue (voir dans l'aperçu des sections), refaites le point 6 de l'exercice 1 en utilisant un Dialogue pour réaliser l'interaction.

Indiquez sur une échelle de 4 la difficulté que vous avez rencontrée à faire cet exercice :
(1 : très facile, 2 : facile, 3 : difficile, 4 : trop difficile)

exercice 2bis : Complétion d'une application simple.



1. Ajouter dans la classe VisageRond la méthode humeur qui admet un paramètre entier pouvant prendre 3 valeurs distinctes (TRANQUILLE, JOYEUX et TRISTE) et fixe l'expressivité du visage. Modifier la méthode « dessiner » en conséquence : Dans l'état tranquille la bouche sera dessinée par un trait horizontal, joyeux par un demi-cercle inférieur et triste par un demi-cercle supérieur.
2. Un ENUM est une **classe particulière** permettant de définir un domaine de valeurs en extension, souvent nommé type énuméré. Il permet de manipuler des valeurs symboliquement (via un nom). Consultez la documentation (Enum type) concernant la définition d'un type ENUM que vous nommerez Humeur et y intégrez les 3 valeurs ci-précédemment citées. Modifiez votre code prendre en compte ce nouveau type.

3. Reprendre la classe VisageRond en construisant une nouvelle classe Oeil pour représenter les caractéristiques d'un œil. Modifier la classe VisageRond pour intégrer 2 objets de cette classe pour représenter les yeux du visage.

4. On souhaite améliorer le mécanisme de rebond du visage. Lorsque le visage atteindra un bord on appliquera les lois du rebond (sans perte d'énergie).

Rendre les codes correspondant aux interventions que vous avez réalisées.

Exercice 3 : compiler et exécuter une applet java

pour en savoir plus sur : le visualiseur d'applets `appletviewer`

Note that you can make it say anything! This text is flowing around the applet because it is left aligned. You can also right align it if you want it to appear on the other side.

- sauvegardez sur votre compte `NervousText.java` le source JAVA de cette applet,
- sauvegardez sur votre compte `essaiapplet.html` un exemple de fichier HTML qui incorpore cette applet,
- compilez la classe `NervousText.java` que vous avez sauvegardé sur votre compte,
- exécutez cette applet
    1. en chargeant le fichier `essaiapplet.html` dans votre navigateur Web
    2. en lançant depuis la ligne de commande la commande `appletviewer essaiapplet.html`
- modifiez `NervousText.java` et/ou `essaiapplet.html` de manière à afficher le texte suivant : "`VIVEMENT QUE NOUS PARTIONS AU SKI !`" (ou tout autre texte de votre choix)

Indiquez sur une échelle de 4 la difficulté que vous avez rencontrée à faire cet exercice :
(1 : très facile, 2 : facile, 3 : difficile, 4 : trop difficile)

Exercice 4 : IDE

Lire la synthèse qui vous est proposée sur les 2 IDEs que sont ECLISPE et NETBEANS dans le document IDE.pdf. Nous préférons très vivement que vous utilisiez le premier d'entre eux.

Lancez l'IDE.

1. Vérifiez que vous utilisez la perspective Java, sinon dans le menu général « Windows » sélectionnez celle-ci.
2. Créez un projet Java de nom Jus.CS310.Tp1 que vous sauvegarderez dans votre répertoire Visages.
3. Créez un package de nom jus.cs310.tp1.
4. Insérez dans ce projet les fichiers source, vous pouvez utiliser la commande import ou le drag&drop.
5. Choisissez la perspective « Resources » et constatez les changements
6. Revenez sur la perspective Java et ajoutez la vue « Navigator » dans le menu « Show view » du menu « Windows »
7. Editez le classpath à partir du menu Project pour y ajouter le jar jus.util.jar.
8. Lancez une exécution de AppliVisage1 avec le menu contextuel de la classe AppliVisage1.java, relancez cette exécution en utilisant l'item AppliVisage1 du menu général run  . Utilisez l'item Run configuration… de ce menu pour changer le nom du lanceur AppliVisage1 en Jus.CS310.Tp1.AppliVisage1. A l'aide du même outil, construisez un nouvel item nommé Jus.CS310.Tp1.Premice qui lance ce programme. Vous indiquerez les arguments nécessaires dans le panneau arguments.

9. Créez un répertoire logique de nom Appli dans le projet, déplacez les sources AppliVisageX dans ce répertoire, constatez l'absence de changement dans la structure de fichiers.
10. Développez les arborescences, constatez que vous accéder aux éléments constitutifs de chaque classe.
11. Détruisez les fichiers .class dans le directory classes.
12. Modifiez l'output directory dans les options du compiler en passant par le menu Project en indiquant le directory classes et recompilez puis testez.
13. changez le classpath en supprimant le directory et en ajoutant le fichier .jar créé précédemment, puis testez.
14. Supprimez le fichier .jar, utilisez dans l'option Build l'option makeJar permettant de créer un tel fichier : indiquez comme cible ..\Visages.jar et construisez le jar qui contiendra VisageRond et Dessin.
15. Créer un folder "docs" et incluez-y les fichiers de documentation que vous souhaitez accessibles, "browsez" un de ceux-ci.

Indiquez sur une échelle de 4 la difficulté que vous avez rencontrée à faire cet exercice :
(1 : très facile, 2 : facile, 3 : difficile, 4 : trop difficile)

Désormais l'ensemble des travaux que vous aurez à réaliser le seront en utilisant systématiquement la structuration en package.

Dans le cadre de la poursuite de ce TP, l'objectif n°1 sera développé dans un sous-package de nom «rakerpak», l'objectif n°2 dans un autre de nom «ferrers» et l'objectif n°3 dans «romain».

Exercice 5 : Algorithmique

Objectif n°1 : Mise en œuvre d'une itération.

L'algorithme suivant consiste à associer à un nombre quelconque *n* un autre nombre *K(n)* généré de la façon suivante :

- On considère les chiffres de *n*, écrits dans une base quelconque (généralement la base 10). On forme le nombre n_1 en arrangeant ces chiffres dans l'ordre croissant et le nombre n_2 en les arrangeant dans l'ordre décroissant.
- On pose $K(n) = n_2 - n_1$.
- On itère ensuite le processus avec *K(n)*.

Pour tout nombre initial, l'algorithme produit au final l'une des possibilités suivantes :

1. 0
2. Un nombre constant
3. Un cycle de nombres

Le cas des nombres entiers en base 10 de 3 chiffres correspond au cas n°1 ou 2.

Ecrire la classe Exercice1 qui possède la méthode de signature « public static int miracle1(int n) ; » qui effectue cet algorithme en imprimant à chaque pas les valeurs caractéristiques pour le cas des nombres à 3 chiffres et renvoie la valeur finale. Pour cela, décomposez le nombre n en un triplet {a2,a1,a0} des chiffres qui le composent, calculez {b2,b1,b0} le triplet représentant n1 et {c2,c1,c0} le triplet représentant n2, vous posez {d2,d1,d0} le triplet représentant n2-n1, puis itérez ce processus tant que {d2,d1,d0} est différent de {a2,a1,a0} ou {0,0,0}. Les a_i,b_i,c_i & $d_i \in$ int.

Exécutez l'algorithme, que constatez-vous ?

1. Montrez formellement quelle valeur a d1 quand a2≠a1≠a0,
2. Montrez formellement quelle relation lie d2 et d0 quand a2≠a1≠a0,
3. Montrez formellement quelles relations lient d2 à a2 et d0 à a0 quand a2≠a1≠a0 :
    a. Dans le cas où l'on considère que a2+a0=9 & a2>a0 & a1=9
    b. Dans le cas où l'on considère que a2+a0=9 & a2<a0 & a1=9

Pour généraliser à des nombres quelconques, nous allons procéder différemment. Ecrire dans la classe Exercice1 les méthodes suivantes :

```
/**
 * retourne le nombre minimal constitué des chiffres de s
 * @param s la suite de chiffres
 * @return le nombre minimal constitué des chiffres de s
 */
 private static int min(String s)
/**
 * retourne le nombre maximal constitué des chiffres de s
 * @param s la suite de chiffres
 * @return le nombre maximal constitué des chiffres de s
 */
private static int max(String s)
/**
 * retourne la séquence de chiffres complétée à gauche par les zéros nécessaires.
 * @param i un nombre
 * @param lg la longueur désirée
 * @return la séquence de chiffres complétée à gauche par les zéros nécessaires.
 */
private static String toString(int i, int lg)

On utilisera un tableau kns de 100 entiers maximum pour stocker la séquence de K(n).
/**
 * retourne l'index de i dans le tableau kns où -1 si celui-ci n'est pas présent
 * @param i un nombre
 * @return l'index de i dans le tableau kns où -1 si celui-ci n'est pas présent.
 */
private static int contains(int i)
```

Objectif n°2 : Mise en œuvre d'un algorithme récursif.

Décomposition d'un nombre n en k parties tel que $n = \sum k_i$.

1. Combien de décompositions existe-t-il si k>n ?
2. Combien de décompositions existe-t-il si k=n ?
3. Dans l'autre condition, pourquoi toute décomposition de n en k partitions peut-elle être décrite à partir de la décomposition de n-1 ou de n-k avec k-1 partitions ou k partitions.

Ecrire dans la classe Exercice2, les méthodes suivantes :

```
/**
 * Restitue la suite des k-partitions de n
 * @param n le nombre à partitionner
 * @param k le nombre de partitions
 * @return la suite des k-partitions de n
 */
public static Suite partition(int n, int k)
/**
 * Restitue la suite des k-partitions de n pour k variant de 1 à n
 * @param n le nombre à partitionner
 * @return la suite des k-partitions de n pour k variant de 1 à n
 */
public static Suite partition(int n)
```

Terminez en réalisant un programme de test et d'affichage. Pour ce faire vous disposez des 2 classes suivantes :

public class **Partition**

Une partition d'un entier dont la valeur est la somme des éléments de la partition

Author: Morat

Constructor Summary

Partition(int i)	Construction une partition ayant i pour seul élément.

Method Summary

void	**addWithOne**() ajoute 1 à chaque élément de la partition.
void	**completeWithOne**() ajoute l'élément 1 à la partition.
String	**toString**() la forme textuelle de la suite.

public class **Suite**

Une suite de partitions

Author: Morat

Constructor Summary

Suite()	Construction d'une suite vide.
Suite(int i)	Construction d'une suite contenant une partition ayant i pour seul élément.

Method Summary

Suite	**concat**(Suite s2) restitue une nouvelle suite concaténation de 2 suites.
Iterator	**iterator**() Restitue un iterator sur l'ensemble des partitions de la suite.
String	**toString**() la forme textuelle de la suite

Objectif n°3 : Mise en œuvre d'une Alternative.

Réaliser un algorithme de conversion entre les représentations décimale et romaine pour des nombres positifs inférieurs à 4000. On rappelle que les valeurs des chiffres romains sont I=1 V=5 X=10 L=50 C=100 D=500 M=1000 et que le rang d'un chiffre romain correspond à sa position dans cette énumération.

Soit $\{R_n, R_{n-1}, .., R_0\}$ un nombre romain, un chiffre romain R tel que Rang(R)=2i répété successivement de 1 à 3 fois vaut de 1 à 3 fois sa valeur, si $Rang(R_{i+1}) > Rang(R_i)$ R_i s'ajoute, si un chiffre romain de rang $Rang(R_{i+1}) < Rang(R_i)$ & $Rang(R_{i+1})$ impair & $0<Rang(R_i)-Rang(R_{i+1})<3$ R_{i+1} se retranche. Toute autre construction est erronée.

1. Quelle est la représentation 1024 en notation romaine ?
2. Quelle est la représentation de 0 en notation romaine ?
3. Quelle est la valeur décimale (ce n'est pas 33 ☺) de MDCLXIV ?
4. La représentation romaine DVD est-elle valide ?
5. Combien au maximum faut-il de caractères pour la représentation romaine de ces nombres ?
6. Combien au maximum faut-il de caractères pour la représentation décimale de ces nombres ?
7. Quel est le taux de rentabilité (n_1/n_2) de la représentation romaine où n_1 est le nombre de représentations valides et n_2 le nombre représentations possibles ?
8. Quel est le taux de rentabilité (n_1/n_2) de la représentation décimale où n_1 est le nombre de représentations valides et n_2 le nombre représentations possibles ?
9. Quel est le taux d'inutilité (n_1/n_2) de la représentation décimale où n_1 est le nombre de représentations inutiles (doublons) et n_2 le nombre représentations valides ?
10. Quel est le taux d'inutilité (n_1/n_2) de la représentation romaine où n_1 est le nombre de représentations inutiles (doublons) et n_2 le nombre représentations valides ?
11. Quelle est la formule de la division euclidienne ?
12. Donnez l'équation récurrente, sur la base de la division euclidienne, du calcul d'un nombre décimal à partir de la valeur des chiffres.

Dans un premier temps on va considérer que les entrées sont correctes. **On utilisera cette seule structure de donnée** :

```
/** l'ensemble des chiffres romains dans l'ordre de leurs rangs */
private static final char[] chiffreRomain = {'I','V','X','L','C','D','M'};
```

Réalisez dans la classe Romain les méthodes suivantes :

```
/**
 * restitue le rang du caractère romain c
 * @param c le caractère romain
 * @return le rang du caractère romain ou -1 si le caractère est incorrect.
 */
private static int rangRomain(char c)
/**
 * restitue la valeur entière d'un chiffre romain,
 * on considère l'entrée correcte.
 * @param c = le chiffre romain
 * @return la valeur entière correspondant au chiffre romain.
 */
private static int romainToDecimal(char c)
/**
 * Restitue la représentation romaine correspondant au chiffre entier i,
 * on considère l'entrée correcte.
 * @param i un chiffre entier à convertir
 * @param p la puissance à laquelle i se trouve
 * @return String la représentation romaine de i*10^p
 */
private static String decimalToRomain(int i, int p){
```

```
/**
 *Restitue la représentation romaine correspondant à la valeur entière i
 * @param i la valeur entière à convertir.
 * @return String la représentation romaine de i
 */
public static String decimalToRomain(int i)
```

Réalisez dans la classe Romain la méthode suivante :

```
/**
 * Restitue la représentation décimale correspondant à la représentation
 * romaine donnée.
 * @param r la représentation romaine
 * @return la représentation décimale.
 */
public static String romainToDecimal(String r)
```

L'algorithme de « romainToDecimal » se fera par un parcours de gauche à droite de la donnée. Vous n'avez le droit d'utiliser que les méthodes suivantes de la classe String :

- charAt(int)
- length()
- substring(int)

Ecrivez un programme principal demandant si l'entrée sera décimale ou romaine, alors saisir l'entrée à convertir et afficher le résultat de cette conversion.

Maintenant on souhaite compléter ce qui vient d'être réalisé pour prendre en compte les erreurs que pourraient comporter les entrées. En cas d'erreur, un message approprié sera affiché, il sera de la forme : "---erreur : ….". Si l'entrée est décimale alors sa valeur doit appartenir à l'intervalle [0,4000[, si l'entrée est romaine la vérification sera faite au cours de la conversion en vérifiant les règles énoncées précédemment. Pour mettre à l'épreuve votre programme, vous utiliserez le programme de test qui vous est fourni et vous indique votre taux de réussite sur le panel des exemples utilisés dans ce test.

Réalisez dans la classe Romain la méthode suivante :

```
/**
 * Réalise la division entière par 10 de la valeur romaine r. Cette
 * transformation se fera sans passer par des valeurs numériques.
 * @param r la représentation romaine d'un nombre [0..4000[
 * @return la représentation romaine r/10
 */
public static String div10(String r)
```

Réalisez dans la classe Exercice3 la méthode suivante :

```
/**
 * Restitue la valeur ⌊log<SUB>10</sub>(r)⌋+1. Ce calcul se fera
 * sans passer par des valeurs numériques.
 * @param r la représentation romaine d'un nombre [0..4000[
 * @return le log à base 10 de la valeur représentée par le nombre romain r.
 */
public static int log10(String r)
```

Reprendre cet exercice afin de pouvoir disposer d'une classe définissant des nombres romains que l'on puisse manipuler. Vous proposerez une méthode d'addition de nombre romain qui rend -1 si l'opération n'est pas réalisable.