



HAUTE ECOLE de la Communauté française en HAINAUT



INSTITUT SUPÉRIEUR INDUSTRIEL À MONS

Techniques Informatiques II

Notes des travaux pratiques (TR2TI2)

Conception et programmation orientées objet avec Java
(Notes provisoires)

GIANNI TRICARICO

Année académique 2011-2012

Table des matières

I	Les bases de la programmation orientée objet	1
	TP1 : Environnement de programmation Java	3
	TP2 : Classes et objets	7
	TP3 : Classes et objets(suite)	11
	TP4 : Tableaux	17
	TP5 : L'héritage	21
	TP6 : Interfaces	27
	TP7 : Le traitement des exceptions)	31
II	Conception IHM en Java - Design patterns	33
	TP1 : Les composants GUI	35
	TP2 : Les composants GUI	36
	TP3 : Les composants GUI	39
	TP1 : Les composants GUI	41
	TP2 : Les composants GUI (suite)	45
	TP2 : Les composants GUI (suite)	48
	TP2 : Les composants GUI (suite)	49
III	Réseaux et objets distribués	51
IV	Annexe	53
A	Correspondances UML - Java	55
	A.1 Classe	55
	A.2 Interface	55
	A.3 Package	56
	A.4 Attribut	56
	A.5 Méthode (opération)	57
	A.6 Héritage (généralisation)	57
	A.7 Réalisation	58
	A.8 Dépendance	59
	A.9 Association	60
	A.10 Agrégation et composition	61

B	La hiérarchie des classes d'exception	63
	Annexe	55
C	Résumé de la gestion des événements	65
D	Ressources	69
	D.1 Liens utiles	69
	D.2 Littératures utiles	69

Première partie

Les bases de la programmation orientée objet

TP1 : Environnement de programmation Java

Durée

1 séance

Objectifs

- Installation du kit de développement Java (JDK)
- Utilisation des outils de ligne de commande
- Utilisation d'un environnement de développement intégré (IDE) : Eclipse
- Manipulation des structures de contrôle

Installation du kit de développement Java

Télécharger le JDK (Java Development Kit) sur le site d'Oracle : <http://www.oracle.com/technetwork/java/index.html>. Sur ce site vous allez rencontrer différents acronymes, voici leur signification :

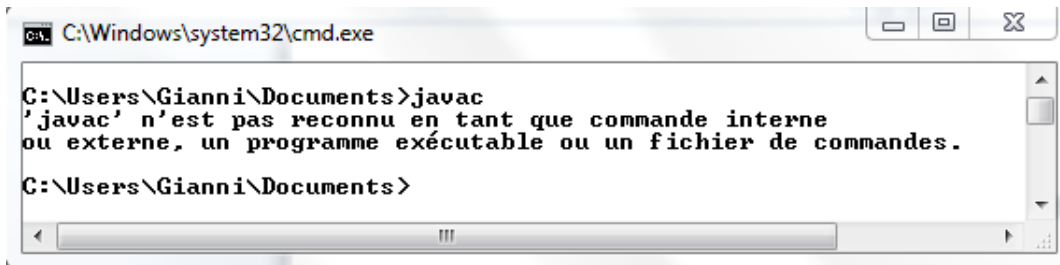
Acronyme	Nom	Explication
Java SE	Standard Edition	La plate-forme Java à utiliser sur les ordinateurs de bureau et les applications de serveur simples
Java EE	Enterprise Edition	La plate-forme Java EE est constituée d'un jeu de services, d'API et de protocoles qui permettent de développer des applications Web
Java ME	Micro Edition	La plate-forme Java à utiliser sur les téléphones portables et autres petits appareils
JRE	Java Runtime Environment	Logiciel destiné aux utilisateurs qui veulent exécuter des applications Java => Environnement d'exécution Java
JDK	Java Development Kit	Logiciel destiné aux développeurs qui souhaitent écrire des applications Java

Lors de l'installation JDK, changez le chemin d'installation par défaut en : `c:\java\jdk1.7.0` (vivement conseillé). La documentation (Java SE 7 Documentation) est contenue dans un fichier compressé séparé du JDK. Vous pouvez la télécharger à l'adresse <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

Remarque : Lors de l'examen, vous n'aurez pas accès à Internet pour obtenir de l'aide. Pour cette raison je vous invite fortement à télécharger cette documentation. Positionnez-vous dans le répertoire `jdk`, dézippez le fichier.

Configurer le chemin d'exécution

Le message d'erreur ci-dessous, signifie que la commande `javac` n'est pas connue dans ce répertoire.



En effet, pour le moment les commandes `javac` et `java` ne sont connues que dans le répertoire `..\jdk1.7.0\bin`. Afin que le système d'exploitation les retrouve, à partir de n'importe quel répertoire, vous devez ajouter le répertoire `..\jdk1.7.0\bin` au chemin d'exécution. La variable d'environnement `path` mémorise la liste des chemins d'exécution. Lorsque l'OS ne trouve pas la commande dans le dossier courant, il parcourt l'ensemble des répertoires repris dans la variable `path` pour la localiser.

Utilisation des outils de ligne de commande

A l'aide de votre éditeur texte favori, recopiez le code source suivant : (Attention à l'encodage des caractères pour les applications consoles! => codage : Européen de l'ouest -> IBM850 ou EOM850 (sur Notepad++))

Listing 1 – Bienvenue.java

```

1 public class Bienvenue
2 {
3     public static void main(String [] args)
4     {
5         System.out.println("Bienvenue en deuxieme annee !");
6     }
7 }

```

Enregistrez ce fichier sous le nom `Bienvenue.java`

Ouvrez une Invite de commandes. Utilisez le menu "Démarrer" et saisissez dans la zone de recherche "cmd". Compilez et exécutez le fichier `Bienvenue.java`

Utilisation d'un environnement de développement intégré (IDE) : Eclipse

Accédez à la page de téléchargement du site Eclipse, puis choisissez **Eclipse IDE for Java Developers**. Décompressez le fichier téléchargé dans un dossier de votre

choix puis lancez Eclipse. Pour faciliter le lancement d'Eclipse créez un raccourci vers le fichier eclipse.exe et placez-le sur le bureau.

Lors du premier lancement, Eclipse propose un dossier par défaut nommé workspace. Créez et sélectionnez plutôt un autre dossier plus personnel, par exemple :

C:\MesTPsJava. Cochez la case "Use this as the default and do not ask again".

Importer des fichiers Java dans Eclipse

1. Vous allez importer un fichier source Java édité plus haut.
2. Après le démarrage d'Eclipse, choisissez File-> New -> Java Project.
3. Saisissez le nom du projet "prjBienvenue".
4. Décochez l'option "Use default location" et indiquez le nom du chemin complet jusqu'au dossier qui contient le fichier Java.
5. Cliquez sur "Finish". Le projet est maintenant créé.

Cliquez sur le triangle situé dans le volet de gauche près du nom du projet (prjBienvenue) pour l'ouvrir, puis cliquez sur le triangle près de "Default package". Double-cliquez sur `Bienvenue.java`. Une fenêtre s'ouvre avec le code source du programme.

Cliquez du bouton droit sur le nom du projet (prjBienvenue), dans le volet le plus à gauche. Sélectionnez Run->Run As->Java Application. La vue Console apparaît au bas de la fenêtre. Le résultat du programme s'y affiche.

Réalisez le même programme avec l'assistant de classe d'Eclipse.

Questions

- 1) Expliquez la démarche à suivre pour définir le chemin d'exécution.
- 2) Que devez-vous spécifier après la commande javac ?
 - a) le nom d'un fichier
 - b) le nom d'une classe
- 3) Que devez-vous spécifier après la commande java ?
 - a) le nom d'un fichier
 - b) le nom d'une classe

Quelques raccourcis clavier utiles

CTRL + Espace : L'indispensable auto-complétion

CTRL + SHIFT + F : Mise en forme du code

CTRL + D : Efface la ligne courante

CTRL + SHIFT + C : pour commenter/décommenter un blocs de lignes

CTRL + SHIFT + L : Affiche la liste des raccourcis clavier

TP2 : Classes et objets

Durée

2 séances

Objectifs

- Utilisation de la classe **Scanner**(lecture d'une entrée au clavier)
- Création de classes et construction d'objets
- Familiarisation avec les règles concernant l'écriture de programmes Java
- Utilisation des arguments de formatage(**String.format**) et des classes **Date** et **GregorianCalendar**
- Utilisation de membres statiques
- Application de la rétention d'information

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice ! Vous pouvez le réaliser soit sur papier, soit à l'aide du logiciel BOUML release 4.22.1.

Exercice 1 : Cahier de notes [2]

- a) Déclaration d'une classe avec une méthode et création d'un objet

Définissez une classe **CahierNotes** contenant une méthode **afficheMessage** affichant à l'écran le message "Bienvenue au cahier de notes!". Pour tester votre classe, définissez une classe principale **CahierNotesTest** quiinstanciera la classe **CahierNotes** et invoquera sa méthode **afficheMessage**.

Donnez la commande pour compiler les deux classes (la plus simple).

- b) Déclaration d'une méthode avec un paramètre et utilisation de la classe **Scanner**

Dans la méthode **main**, invitez l'utilisateur à saisir le nom du cours à l'aide de la classe **Scanner**.

Modifiez la méthode `afficheMessage` pour qu'elle affiche le message de bienvenue suivi du nom du cours. Cette nouvelle méthode requière un paramètre qui représente le nom du cours.

c) Variables d'instances, les méthodes *set(mutateur)* et les méthodes *get(accesseur)*

La classe `CahierNotes` va contenir une variable d'instance `nomCours`(private) permettant de stocker le nom du cours. Elle contiendra également trois méthodes - `setNomCours`, `getNomCours` et `afficheMessage`. La méthode `setNomCours` stockera le nom du cours dans la variable d'instance `nomCours`. La méthode `getNomCours` permettra d'obtenir le nom du cours. La méthode `afficheMessage`, sans paramètre, affichera le message de bienvenue suivi du nom du cours.

La classe `CahierNotesTest` devra permettre :

- d'afficher la valeur initiale de la variable `nomCours`,
- de demander à l'utilisateur d'introduire le nom du cours,
- d'exécutez la méthode `afficheMessage`.

Dessinez le diagramme UML des deux classes !

d) Constructeur

Ajoutez un constructeur à la classe `CahierNotes` pour initialiser l'attribut `nomCours` à « TR2TI2 – Techniques Informatiques 2 ».

Dessinez le diagramme UML des deux classes !

Exercice 2 : Compte bancaire

Implémentez la classe `CompteBancaire` permettant de stocker le solde d'un compte bancaire. La variable d'instance `solde` sera de type `double`. Cette classe possède un constructeur et deux méthodes. A la création d'un compte, le programmeur doit préciser le solde initial. Attention, pour un montant négatif, le solde sera initialisé à 0.0. Une méthode `depot` permet d'ajouter un montant sur le compte. Une méthode `getSolde` permet de récupérer le solde.

Implémentez la classe `CompteBancaireTest` permettant de tester la classe `CompteBancaire`. Cette classe créera deux comptes avec un montant initial de 50.00 Euro et de -7.50 Euro respectivement. Puis l'application affichera l'état initial des comptes. Ensuite, l'application déposera un montant sur le premier compte et le solde des comptes sera affiché. Pour finir, l'application déposera un montant sur le deuxième compte et le solde des comptes sera affiché.

Exercice 3 : Quelle est votre fréquence cardiaque cible ? [2]

Référence : http://www.becel.ca/fr_ca/gerer_sa_sante_du_coeur/lactivite_physique/quelle_est_votre_frequence_cardiaque_cible.aspx

Pendant l'exercice, il existe une zone de fréquences cardiaques optimales dénommée zone cible. En veillant à demeurer dans cette zone, vous seriez plus en mesure de tirer les bienfaits en courant moins de risques pour votre santé. Pour tirer le maximum de bienfaits tout en minimisant les risques, vous devriez viser entre 60 et 80 % de votre fréquence cardiaque maximale lorsque vous faites de l'exercice.

- a) Commencez en estimant votre fréquence cardiaque maximale (FCM). Pour ce faire, soustrayez votre âge de 220. Par exemple, si vous avez 40 ans, votre fréquence cardiaque maximale est de 180 ($220 - 40 = 180$).
- b) Ensuite, calculez votre zone pour maintenir une fréquence cardiaque cible en multipliant votre FCM par 0,6 et par 0,8. Par exemple, si votre FCM est de 180, votre zone se situe entre 108 et 144 battements par minute ($180 \times 0,6 = 108$ et $180 \times 0,8 = 144$).

Créez une classe, nommée **CardioFreq**, qui contiendra les attributs suivants :

- le nom et le prénom de la personne,
- la date de naissance de la personne (séparer la date en trois attributs : jour, mois et année).

Elle contiendra un constructeur qui recevra ces informations en paramètres.

De plus, elle possèdera les méthodes suivantes :

- une méthode qui calcule et retourne l'âge de la personne (année),
- une méthode qui calcule et retourne la fréquence cardiaque maximale (FCM),
- une méthode qui calcule et retourne la fréquence cardiaque cible

Pour chaque attribut, définissez un accesseurs (méthode `get`) et un mutateurs (méthode `set`).

Ecrivez une application qui invite l'utilisateur à introduire son nom, son prénom et sa date de naissance. Elleinstanciera un objet de type **CardioFreq** et affichera à l'écran les informations suivantes :

- le nom, le prénom et l'âge de la personne,
- la fréquence cardiaque maximale,
- la zone cible.

Exemple d'affichage :

Nom : toto Prénom : titi Date : 6/4/1984

FCM : 193

Fréquence cible : 116 - 154

Exercice 4 : Factures

Définir une classe Facture qui contient les attributs suivants :

- Taux de la TVA (constante)
- Numéro facture (auto-incrémenté)
- Montant HT

Définissez un constructeur qui accepte comme paramètres formels : montant HT (vérifier la validé du montant)

Définissez une méthode (privée ? ou publique ?) `ValideMontant(float montant)` dont l'objectif est de contrôler si le montant passé en paramètre est strictement positive.

Définissez un mutateur `setMontant` chargé de modifier le montant HT d'une facture. On devra aussi vérifier que le nouveau montant est strictement positif.

TP3 : Classes et objets(suite)

Durée

3 séances

Objectifs

- Utilisation package
- Énumération et classe `Random`
- Surcharge de méthode
- Clonage d'objet
- Destruction d'objet par le garbage collector

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice !

Exercice 1 : jeu de craps [2]

Références : <http://www.argentgagnant.com/craps/jeu-craps.html>, <http://en.wikipedia.org/wiki/Craps>

Le craps se joue avec 2 dés.

Déroulement du jeu de craps :

Le joueur lance les dés

- S'il fait 7 ou 11, il gagne sa mise.
- S'il fait 2, 3 ou 12, il perd sa mise. On dit qu'il a fait un craps.
- S'il fait 4, 5, 6, 8, 9 ou 10, on dit que le joueur établit le point et le croupier pose un palet sur la table pour indiquer la valeur du point sorti aux dés.

Une fois le point établi, le joueur relance les dés jusqu'à refaire le point ou à tirer 7.

- S'il refait le point établi, il gagne sa mise.

- S'il fait 7 avant d'avoir refait le point, il perd sa mise.

Lexique du craps :

- Natural : 7 (5+2, 6+1, ...)
- Yo-leven : 11 (5+6 ou 6+5)
- Snake Eyes : 2 (1+1)
- Ace Deuce : 3 (2+1 ou 1+2)
- Boxcars : 12

Définissez une énumération pour l'état du joueur : `GAGNE`, `PERDU` et `CONTINU`. Définissez des constantes pour les noms des figures : `NATURAL`, `YO_ELEVEN`, `SNAKE_EYES`, `ACE_DEUCE` et `BOXCARS`.

L'application doit simuler le jeu de craps, en utilisant des méthodes implémentant la logique du jeu. Dans la méthode `main` de la classe `CrapsTest`, créez un objet de la classe `Craps` et appelez sa méthode `play` pour démarrer le jeu. Définissez et implémentez la classe `De` (attributs ? et méthodes ?). Pour simuler le lancer du dé, utilisez la classe `Random`. Placez les classes dans le package `be.isims.coo.tp3.ex1`.

Exercice 2 : nombre complexe

Définissez et implémentez une classe `Complexe` pour réaliser des opérations arithmétiques sur les nombres complexes.

$$\text{nbreComplexe} = \text{partieRéelle} + \text{partieImaginaire} * i$$

Ecrivez un programme pour tester votre classe. Utilisez des variables réelles représentant les données privées de la classe. Implémentez un constructeur permettant d'initialiser le nombre complexe. Dans le cas où il n'est pas initialisé, prévoyez des valeurs par défaut.

Implémentez les méthodes qui réalisent les opérations suivantes :

- Additionner deux nombres complexes
- Soustraire deux nombres complexes
- Afficher les nombres complexes sous la forme (a,b) où a est la partie réelle et b la partie imaginaire.

Qu'entend-on par surcharge d'opérateur ?

Est-ce que le java intègre ce mécanisme ?

Exercice 3 : nombres rationnels

Implémentez une classe afin de représenter des nombres rationnels. Un nombre rationnel comporte un numérateur et un dénominateur, tous deux de type entier. Puisque chaque nombre possède son propre numérateur et dénominateur, ces variables sont des variables d'instance.

La classe `Rationnel` possède deux constructeurs. L'un d'eux a deux paramètres formels, dont les valeurs servent pour l'initialisation des variables d'instance. L'autre constructeur n'a qu'un paramètre, il s'agit du numérateur ; la valeur du dénominateur est alors 1.

Implémentez des méthodes d'accès, en lecture seulement, afin de retourner la valeur du numérateur et la valeur du dénominateur respectivement.

Implémentez la méthode d'instance `plus`. Cette méthode n'a qu'un paramètre formel, de type `Rationnel`. La méthode retourne un nouvel objet de type `Rationnel` qui représente la somme de ce nombre et celui passé en paramètre.

Implémentez une méthode de classe privée afin de calculer le plus grand diviseur commun de deux nombres, passés en paramètres. Dessinez l'organigramme (LARP) de l'algorithme utilisé pour le calcul PGDC.

Implémentez une méthode d'instance privée que vous nommerez `formeReduite` et qui transforme cette instance en sa forme réduite.

Effectuez tous les changements nécessaires afin que ce nombre soit toujours sauvegardé sous sa forme réduite.

Implémentez une méthode `public String toString()` qui retourne une chaîne de caractères représentant cette fraction. Le format sera numérateur suivi de « / » suivi du dénominateur, ou encore, simplement le numérateur, si le dénominateur est 1.

Exercice 4 : Vie et mort des objets [1]

Cet exercice met en évidence la création et la destruction des objets dans la mémoire tas.

Implémentez la classe `Classe1` qui possède un attribut faisant référence à la classe `Classe2` et qui crée dans son constructeur une instance de cette classe. Chaque classe possédera un constructeur affichant le message « Un nouvel objet de type Classe1 est créé » pour la `Classe1` et le message « Un nouvel objet de type Classe2 est créé » pour la `Classe2`. Chaque classe possédera une méthode `protected void finalize()`. Cette méthode s'exécute lorsque l'objet est détruit. Cette méthode affichera le message « L'objet de type Classe1 se meurt ... » pour la `Classe1` et le message « L'objet de

type `Classe2` se meurt ... » pour la `classe2`.

Implémentez la classe `VieMortObjet` permettant de créer une instance de la classe `Classe1`. Mettez en œuvre, dans cette classe, trois situations dans lesquelles les objets seront détruit explicitement par le ramasse-miette.

Remarque : L'appel explicite du garbage-collector se réalise à l'aide de l'instruction suivante : `System.gc()` ;

Exercice 5 : Encapsulation [5]

Observez et testez le code ci-dessous :

Listing 2 – `Employe.java`

```
1 import java.util.Date;
2 import java.util.GregorianCalendar;
3
4 public class Employe {
5
6     private String nom;
7     private Date dateNaissance;
8
9     public Employe(String nom,int jour,int mois,int annee)
10    {
11        this.nom=nom;
12        GregorianCalendar calendrier=new GregorianCalendar(annee,
13            mois-1,jour);
14        dateNaissance= calendrier.getTime();
15    }
16    public Date getDateNaissance()
17    {
18        return dateNaissance;
19    }
20 }
```

Listing 3 – `EmployeTest.java`

```
1 import java.util.Date;
2
3 public class EmployeTest {
4
5     public static void main(String[] args) {
6         Employe employee=new Employe("toto",23,7,1990);
7         Date dateNaissance= employee.getDateNaissance();
8         System.out.println(dateNaissance);
9         dateNaissance.setYear(100);
10        dateNaissance= employee.getDateNaissance();
11        System.out.println(dateNaissance);
12    }
13
14 }
```

Quel est le problème de la classe `Employe` ?

Comment allez-vous le corriger ?

TP4 : Tableaux

Durée

3 séances

Objectifs

- Déclaration et création de tableau
- Initialisation des éléments d'un tableau
- Génération de fichier JAR exécutable

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice !

Package : `be.isims.coo.tp4.ex#`

Exercice 1 : Cahier de notes [2]

Ecrivez une classe `CahierNotes` qui utilise un tableau d'entiers pour stocker l'ensemble des notes obtenues par les étudiants lors d'examen. Le constructeur à deux paramètres – le nom du cours et le tableau de notes. Quand l'application (classe `CahierNotesTest`) crée un objet de type `CahierNotes`, l'application passe un tableau d'entiers au constructeur, qui affecte la référence du tableau à la variable d'instance `notes`. La taille du tableau `notes` est déterminée par la longueur du tableau passé au constructeur. Ainsi, l'objet de type `CahierNotes` peut traiter un nombre quelconque de notes.

Dans l'application de test, vous initialiserez un tableau avec des valeurs de notes. La méthode `traitementNotes` contient une série d'appel de méthodes qui affichera un rapport comprenant les notes. L'appel de la méthode `AfficherNotes` affiche la liste des étudiants avec leurs notes respectives. La méthode `traitementNote` appelle ensuite la méthode `getMoyenne` pour obtenir la moyenne des notes (utilisez l'instruction `for` de la forme `for (:)`). La méthode `traitementNote` appelle les méthodes `getMinimun` et `getMaximum` pour déterminer la plus basse et la plus haute note. Finalement la méthode `traitementNote` appelle la méthode `sortieHistogramme` pour afficher la répartition graphique des notes (voir ci-dessous).

Bienvenue au cahier de note du cours : Info

Voici les notes sur 100 obtenues par les étudiants :

Etudiant 1 : 87

Etudiant 2 : 68

Etudiant 3 : 94

Etudiant 4 : 100

Etudiant 5 : 83

Etudiant 6 : 78

Etudiant 7 : 85

Etudiant 8 : 91

Etudiant 9 : 76

Etudiant 10 : 87

La moyenne de la classe : 84,90

La note la plus basse : 68

La note la plus haute : 100

Répartition des notes :

0- 9:

10-19:

20-29:

30-39:

40-49:

50-59:

60-69:*

70-79:**

80-89:****

90-99:**

100:*

Exercice 2 : Mélanger un paquet de cartes et les distribuer

Réalisez une application qui :

- construit un paquet de cartes
- mélange le paquet
- distribue les cartes
- affiche les cartes distribuées

Commencez par le diagramme de classes avant de rédiger le programme!!!!
Ensuite, faites valider votre diagramme auprès de votre enseignant.

Exercice 3 : Carré magique [6]

En mathématiques, un carré magique d'ordre n est composé de n^2 nombres entiers. Généralement distincts et écrits sous la forme d'un tableau carré. Ces nombres sont disposés de manière à ce que leurs sommes sur chaque rangée, sur chaque colonne et

sur chaque diagonale principale soient égales. Dans cet exercice, vous écrirez une application pour déterminer si un carré est magique ou non.

La classe `Carre` contient le code qui représente une matrice carrée. Elle contient un constructeur qui requière un paramètre représentant l'ordre du carré et les méthodes pour lire les valeurs du carré. Imprimer le carré, calculer la somme d'une rangée donnée, calculer la somme d'une colonne donnée, calculer la somme de la diagonale principale et déterminer si le carré est magique ou non. Vous disposez de la méthode `lectureCarre`, à vous d'écrire les autres. Notez que la méthode `lectureCarre` possède un paramètre de type `Scanner`. La classe `CarreTest` contient le code lisant les carrés à partir d'un fichier nommé `magicData.txt` et permettant d'affirmer s'ils sont magiques ou non. Notez que la méthode principale lit la taille du carré et appelle la méthode `lectureCarré` pour lire le carré.

Voici comment utiliser la classe `Scanner` pour lire le fichier :

```
Scanner scan = new Scanner(new File("magicData.txt"));
```

Notez que dans le fichier (`magicData.txt`), le -1 en bas indique au programme d'arrêter la lecture. Importez le projet `prjCarreMagique` dans Eclipse.

Exercice 4 : Fichier JAR

Créez un fichier d'archive exécutable (JAR exécutable) pour chacun des projets précédents. Exécutez vos applications à partir des fichiers jar.

Comment avez-vous créé vos fichiers jar sous Eclipse ?

Quelle commande avez-vous tapé dans la console pour les exécuter ?

TP5 : L'héritage

Durée

3 séances

Objectifs

Mise en oeuvre des techniques de base de l'héritage :

- Redéfinition de méthode
- Polymorphisme
- Liaison dynamique
- Classes et méthodes abstraites (rendre effectif)
- Utilisation du transtypage et de l'opérateur `instanceof`

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice !

Package : `be.isims.coo.tp5.ex#`

Exercice 1 : Carré

Implémentez une sous-classe `Carre` qui étend la classe `Rectangle`. Son constructeur possède les positions X et Y du centre et la longueur du côté du carré. Appelez les méthodes `setLocation` et `setSize` de la classe `Rectangle`. Cherchez ces méthodes dans la documentation de la classe `Rectangle`. Définissez une méthode `getArea` qui calcule et retourne la surface du carré. Ecrivez un programme qui demande le centre et la longueur du côté du carré, qui affiche les valeurs (X,Y et côté) du carré (utilisez la méthode `toString` héritée de la classe `Rectangle`) et de la surface du carré.

Exercice 2 : L'horloge

Procurez-vous le fichier `LibClock.jar` et la documentation associée auprès de votre enseignant. Lisez attentivement la documentation de cette classe.

a) Utilisation de la classe existante : `horloge.Clock`

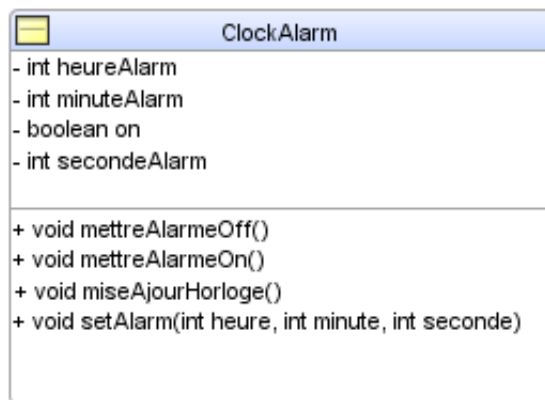
N'oubliez pas de référencier le package `LibClock.jar` dans votre projet. Pour vérifier si ce lien est correct, écrivez un programme construisant deux horloges. Changez la couleur de fond d'une horloge.

b) Etendre une classe existante

Créez une classe `ClockAlarm` qui hérite de la classe `Clock`. Cette nouvelle classe permettra d'obtenir une horloge équipée d'une alarme.

Une méthode permettra de couper l'alarme `mettreAlarmeOff()` et une autre de l'activer `mettreAlarmeOn()`.

Le paramétrage de l'heure de l'alarme se fera à l'aide de la méthode `setAlarm()`.



Exercice 3 : tableaux dynamiques polymorphes

Réalisez une application utilisant une collection `ArrayList` pour y stocker différents objets. Voici la liste des objets à ajouter dans cette `ArrayList` :

- un dé (classe définie dans les TP précédents),
- une carte (classe définie dans les TP précédents) ,
- un rectangle (classe prédéfinie `java.awt.Rectangle` qui se trouve dans l'API de Java),
- une chaîne de caractères (classe prédéfinie `String` qui se trouve dans l'API de Java)
- .

Parcourez cette collection avec une boucle `for each` afin de récupérer les objets pour accéder à leurs méthodes. Voici les actions à réaliser en fonction de l'objet en cours :

- pour l'objet de type `De`, lancer le dé et afficher sa valeur
- pour l'objet de type `Carte`, afficher la couleur et la valeur de la carte

- pour l'objet de type **Rectangle**, afficher les caractéristiques du rectangle (**toString**)
- pour l'objet de type **String**, afficher la chaîne de caractères

Quelles opérations avez-vous dû réaliser avant de pouvoir manipuler l'objet sorti de la collection ?

Exercice 4 : Calcul de la masse salariale [2]

La masse salariale désigne la somme des salaires que paie une entreprise. Vous allez programmer le calcul de la masse salariale hebdomadaire d'une entreprise. Cette entreprise comporte plusieurs types d'employés :

- des salariés (**Salarie**) qui sont payés suivant un barème fixe quelque soit le nombre d'heures de travail
- des employés (**SalarieHoraire**) qui sont payés suivant le nombre d'heures de travail dans la semaine. Ils sont payés à un certain tarif horaire et leurs heures supplémentaires (au-delà de 38 heures) sont payées 30 % de plus que les heures normales
- des vendeurs (**Vendeur**) sont payés uniquement à la commission. Elle correspond à un pourcentage (taux de commission) des ventes.
- des vendeurs (**VendeurPlusBase**) reçoivent un salaire de base plus une commission (même taux que les vendeurs du dessus)

Modélisez cette situation à l'aide du diagramme de classes. Faites vérifier par l'enseignant avant de commencer à implémenter les classes.

Chaque employé, indépendamment de la manière dont ses revenus sont calculés, a un prénom, un nom de famille et un numéro de Sécurité Social. De plus, il possède une méthode **revenus** et une méthode **toString**.

Type d'employé	Méthode toString
Salarie	Employé salarié : nom prénom Numéro de sécurité sociale : NSS Salaire hebdomadaire : xxx Euro
SalarieHoraire	Employé horaire : nom prénom Numéro de sécurité sociale : NSS Salaire horaire : xxx Euro/h Heures prestées :
Vendeur	Vendeur : nom prénom Numéro de sécurité sociale : NSS Ventes : xx Euro Taux de commission : .0.0
VendeurPlusBase	Vendeur plus base : nom prénom Numéro de sécurité sociale : NSS Ventes : xx Euro Taux de commission : .0.0 Salaire de base : xxEuro

L'application crée un objet pour chacune des quatre classes. Le programme manipule ces objets de façon non polymorphique, via des références du type de l'objet.

Puis le programme manipule ces objet de façon polymorphe en utilisant un tableau statique de type générique. En traitant les objets, le programme augmente le salaire de base de chaque vendeur (**VendeurPlusBase**) de 10 %. Ceci exige la détermination du type de l'objet au cours de l'exécution.

Exercice 5 : Améliorer l'extensibilité d'une application [3]

Voici l'application que vous devez améliorer :

Listing 4 – FormeTest.java

```

1 package be.isims.tp5;
2
3 import java.util.ArrayList;
4
5 public class FormeTest {
6
7     public static void main(String[] args) {
8         ArrayList<Shape> listeFormes = new ArrayList<Shape>();
9         listeFormes.add(new Shape(TypeForme.CARRE));
10        listeFormes.add(new Shape(TypeForme.CARRE));
11        listeFormes.add(new Shape(TypeForme.CERCLE));
12        ZoneDessin zone=new ZoneDessin();
13        zone.DessinerToutesFormes(listeFormes);
14    }
15 }
```

Listing 5 – Shape.java

```
1  /**
2   *
3   */
4  package be.isims.tp5;
5
6  public class Shape {
7
8      private TypeForme forme;
9
10     public Shape(TypeForme forme) {
11         this.forme = forme;
12     }
13
14     TypeForme getForme() {
15         return forme;
16     }
17
18 }
```

Listing 6 – TypeForme.java

```
1  package be.isims.tp5;
2
3  public enum TypeForme {CARRE, CERCLE};
```

Listing 7 – ZoneDessin.java

```
1  package be.isims.tp5;
2
3  import java.util.ArrayList;
4
5  public class ZoneDessin {
6
7      public void DessinerToutesFormes(ArrayList<Shape> listeFormes) {
8          for (Shape forme : listeFormes) {
9              switch (forme.getForme()) {
10                 case CARRE:
11                     dessinerCarre(); break;
12                 case CERCLE:
13                     dessinerCercle(); break;
14             }
15         }
16     }
17
18     private void dessinerCarre() {
19         System.out.println("Je dessine un carre");
20     }
21
22     private void dessinerCercle() {
23         System.out.println("Je dessine un cercle");
24     }
25
26
27 }
```

28 }

Dans un premier temps, testez cette application. Imaginons que l'on souhaite ajouter une nouvelle forme, par exemple un `triangle`. Vu la structure actuelle, vous devez modifier la classe `ZoneDessin` et l'énumération `TypeForme` !

Il est fortement déconseillé de modifier une classe qui fonctionne correctement, vous risquerez d'ajouter un bug !

De plus, imaginons que cette classe soit "fermée" (compilée). Vous n'avez donc pas accès au code source de celle-ci !

Repensez à la structure de l'application pour qu'elle soit plus facilement extensible. Montrez avec votre nouvelle architecture, qu'il est possible d'ajouter **x formes** sans devoir modifier la classe `ZoneDessin`.

TP6 : Interfaces

Durée

3 séances

Objectifs

- Réutiliser des fonctionnalités prédéfinies dans l'API Java
- Rendre les classes réutilisable

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice !

Package : `be.isims.coo.tp6.ex#`

Exercice 1 : Interface pour la réutilisation de classe [4]

Réalisez une classe `CollectionDonnees` qui permet de calculer la moyenne et le maximum d'un ensemble de nombres entiers. La méthode `add` permet d'ajouter des nombres. La méthode `getMoyenne` renvoie la moyenne des données et la méthode `getMaximum` retourne le nombre le plus grand. Remarque : Ne pas utiliser de tableau pour stocker les nombres.

Nous souhaiterons utiliser les services de la classe `CollectionDonnees` dans d'autres contextes. Par exemple, nous voulons traiter des comptes bancaires pour trouver celui possédant le solde le plus élevé, nous ne pouvons pas utiliser la classe dans sa forme actuelle. Nous pourrions modifier la classe, comme ceci :

```
public class CollectionDonnees
{
    private double somme;
    private CompteEnBanque maximum;
    . . .
    public void add(CompteEnBanque x)
    {
        somme = somme + x.getSolde();
        . . .
    }
}
```

```
    }  
    public CompteEnBanque getMaximum()  
    {  
        return maximum;}  
}
```

Un autre exemple, supposons que nous voulons trouver la pièce de monnaie avec la valeur la plus élevée parmi un jeu. Nous pourrions modifier la classe, comme ceci :

```
public class CollectionDonnees  
{  
    private double somme;  
    private Coin maximum;  
    . . .  
    public void add(Piece x)  
    {  
        somme = somme + x.getValue();  
    . . .  
    }  
    public Piece getMaximum()  
    {  
        return maximum; }  
}
```

Comment pouvez-vous rendre la classe `CollectionDonnees` plus générique (acceptant n'importe quels objets sur lesquels on peut appliquer une moyenne et déterminer la valeur la plus élevée) ?

Donnez dans un premier temps, la solution sous la forme d'un diagramme de classe. Pensez aux interfaces. Faites-le vérifier par l'enseignant avant de commencer à implémenter les classes.

Exercice 2 : Utilisation du composant Timer [5]

Le package `javax.swing` contient une classe `Timer`, utile pour être averti de l'expiration d'un délai imparti.

A la construction d'un minuteur, vous définissez l'intervalle de temps et lui indiquez ce qu'il doit faire lorsque le délai est écoulé.

Le minuteur doit savoir quelle méthode appeler. Vous devez spécifier un objet d'une classe qui implémente l'interface `ActionListener` du package `java.awt.event`. Voici la définition de cette interface :


```
public interface ActionListener
{
    void actionPerformed(ActionEvent event); <<-----
}
```

Le minuteur appelle la méthode `actionPerformed` lorsque le délai est expiré.

Supposons que vous souhaitiez afficher le message "Au bip, il sera ...", suivi d'un bip, et ce toutes les 10 secondes. Définissez une classe `AffichageMinute` qui implémente l'interface `ActionListener`. Vous placerez alors toutes les instructions que vous voulez voir exécuter dans la méthode `actionPerformed`. Pour réaliser un bip, voici l'instruction :

```
Toolkit.getDefaultToolkit().beep();
```

Construisez ensuite un objet de cette classe dans la classe principale `MinuteurTest` et transmettez-le au constructeur du `Timer`. Enfin, vous démarrez le minuteur.

Exercice 3 : Tri personnalisé `Array.sort(Object[] a)` [5]

La méthode `sort` de la classe `Arrays` permet de trier un tableau d'objets. Mais à vous de définir votre *critère de tri* par l'implémentation de l'interface `Comparable` et l'obligation de rendre effective la méthode `compareTo` que cette dernière possède.

```
public interface Comparable
{
    int compareTo(Object other);
}
```

Utilisez la méthode `Array.sort` pour trier un tableau d'objets `Employe`. Supposons que nous voulions comparer le salaire des employés. Testez votre classe `Employe`.

```
class Employee
{
    private String nom;
    private double salaire;
    public Employee(String n, double s)
    {
        nom = n;
        salaire = s;
    }
    public String getNom()
    {
        return nom;
    }
    public double getSalaire()
```

```
{  
    return salaire;  
}
```

Depuis Java SE 5.0, l'interface `Comparable` a été améliorée pour devenir un type générique :

```
public interface Comparable<T>  
{  
    int compareTo(T other); // le paramètre a le type T  
}
```

Quelle est la différence entre ces deux versions d'interface ?

TP7 : Le traitement des exceptions

Durée

1 séances

Objectifs

- Signaler les exceptions sous contrôle
- Comment lancer une exception
- Créer des classes d'exception
- Capturer les exceptions

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice !

Package : `be.isims.coo.tp7.ex#`

Exercice 1 : Saisie incorrecte et opération arithmétique non conforme

Réalisez une application invitant l'utilisateur à saisir deux entiers et les transmet à la méthode `quotient`, qui calcule la division entière et retourne le quotient.

Dans le cas d'une saisie incorrecte, le programme réinvite l'utilisateur à saisir un entier. Après trois essais infructueux, l'application se termine. Ce même mécanisme sera d'application pour le cas d'une division par zéro sauf que le nombre d'essais sera fixé à cinq.

Quittez l'application à l'aide `System.exit`.

Testez votre code en simulant les exceptions.

Exercice 2 : Compte bancaire

Reprenez l'exercice 2 du TP 2 et ajoutez à la classe `CompteBancaire` une méthode `retrait`.

Lors du retrait si le montant retiré est supérieur au solde du compte, lancer une exception du type `IllegalArgumentException`. Traitez ensuite cette exception en affichant le message "Solde insuffisant!".

Modifiez la méthode retrait de telle manière quelle lance une exception personnalisée nommée `SoldeInsuffisantException` basé sur `IllegalArgumentException`. Cette nouvelle exception, affichera le message "Le retrait de x euros excède votre solde qui est de x euros". Puis, interceptez cette exception et affichez son message.

Deuxième partie

Conception IHM en Java - Design patterns

TP1 : Design pattern Composite

Objectifs

Mise en œuvre du design pattern Composite.

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice!!!

Package : `be.isims.ihm.tp1.ex#`

Exercice 1 :

Ecrire le schéma UML représentant des expressions binaires à l'aide du patron Composite. b) Ecrire les classes correspondantes permettant d'évaluer ces expressions. Une expression binaire se définit ici par : `<expbin> = <expbin> + <opérateur> + <expbin>` ou `<nombre>`. Les nombres sont des entiers. Le seul opérateur utilisé est l'addition.

Exercice 2 :

Supposons que vous devez construire un logiciel pour un revendeur connu. Ce dernier possède plusieurs magasins répartis dans tout le pays. L'application doit être capable de calculer le bénéfice par ville et par province.

Exercice 3 :

Un système de fichiers contient des fichiers et des répertoires. Ces répertoires contiennent à leur tour des fichiers et des sous-répertoires. On veut pouvoir lister le système de fichiers, les répertoires et les fichiers (en donnant leur nom).

TP2 : Design pattern Strategy

Objectifs

Mise en œuvre du design pattern stratégie.

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

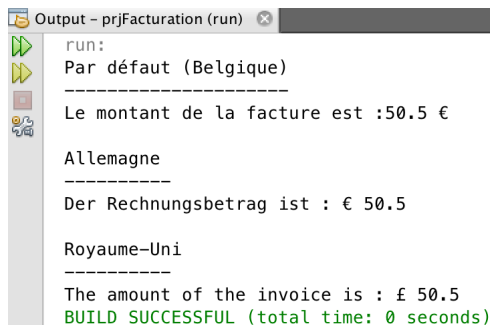
Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice!!!

Package : be.isims.ihm.tp2.ex#

Exercice 1 :

On se place dans le cadre d'une application de facturation destinée à plusieurs pays. Proposez une solution flexible permettant de personnaliser l'affichage d'une facture selon le pays. Le texte et les prix doivent être automatiquement adaptés. La solution proposée doit faciliter l'ajout d'un nouveau pays sans changer l'interface du client. Les pays visés sont la Belgique, le Royaume-Uni et l'Allemagne. Modélisez puis programmez les différents algorithmes nécessaires à la mise en place de ce système.



```

Output - prjFacturation (run)
run:
Par défaut (Belgique)
-----
Le montant de la facture est :50.5 €

Allemagne
-----
Der Rechnungsbetrag ist : € 50.5

Royaume-Uni
-----
The amount of the invoice is : £ 50.5
BUILD SUCCESSFUL (total time: 0 seconds)
  
```

```
package prjfacturation;
```

```
public class Client {
```

```

    public static void main(String[] args) {
        Facture maFacture=new Facture(50.50);
        System.out.println("Par défaut (Belgique)");
        System.out.println("-----");
    }
}
  
```



```

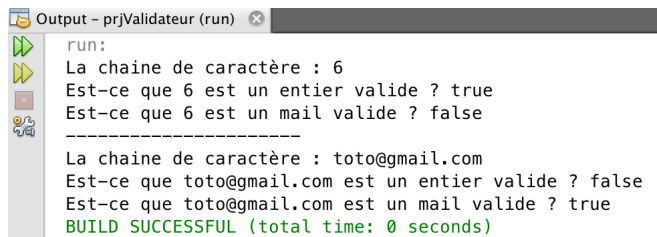
        maFacture.affiche();

        System.out.println("\nAllemagne");
        System.out.println("-----");
        maFacture.setAffichage(new AffichageAllemagne());
        maFacture.affiche();
        System.out.println("\nRoyaume-Uni");
        System.out.println("-----");
        maFacture.setAffichage(new AfficheRoyaumeUni());
        maFacture.affiche();
    }
}

```

Exercice 2 :

On veut définir une classe *Valideur* capable de valider différentes saisies (sous forme de *String*), par exemple la saisie de valeurs entières ou la saisie d'adresse mail. Pour la validation d'entiers, utiliser la méthode *Integer.parseInt(chaine)*. Pour le mail, utiliser la méthode *indexOf()* de la classe *String*.



```

Output - prjValideur (run)
run:
La chaine de caractère : 6
Est-ce que 6 est un entier valide ? true
Est-ce que 6 est un mail valide ? false
-----
La chaine de caractère : toto@gmail.com
Est-ce que toto@gmail.com est un entier valide ? false
Est-ce que toto@gmail.com est un mail valide ? true
BUILD SUCCESSFUL (total time: 0 seconds)

```

```
package prjvalideur;
```

```
public class Client {
```

```

    public static void main(String[] args) {
        String txt="6";
        System.out.println("La chaine de caractère : "+txt);
        Valideur valEntier=new Valideur(new FormatEntier(),txt);
        System.out.println("Est-ce que "+txt+ " est un entier valide ? "+valEntier.isV

        Valideur valMail=new Valideur(new FormatMail(),txt);
        System.out.println("Est-ce que "+txt+ " est un mail valide ? "+valMail.isValid

        System.out.println("-----");

        txt="toto@gmail.com";
        System.out.println("La chaine de caractère : "+txt);
    }
}

```

```
    Validateur valEntier2=new Validateur(new FormatEntier(),txt);
    System.out.println("Est-ce que "+txt+ " est un entier valide ? "+valEntier2.isVali

    Validateur valMail2=new Validateur(new FormatMail(),txt);
    System.out.println("Est-ce que "+txt+ " est un mail valide ? "+valMail2.isVali

}

}
```

TP3 : Design pattern Observer

Objectifs

Mise en œuvre du design pattern observateur.

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

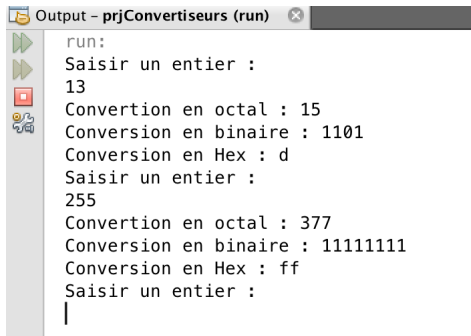
Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice!!!

Package : be.isims.ihm.tp2.ex#

Exercice 1 :

Un message sur la console demande de saisir un entier (de manière répétitive dans une boucle infinie). A chaque saisie, un objet observé contenant la valeur est mis à jour. Trois observateurs de cet objet sont notifiés, et affichent sur la console les conversions hexadécimale, octale et binaire du nombre saisi.



```
run:
Saisir un entier :
13
Conversion en octal : 15
Conversion en binaire : 1101
Conversion en Hex : d
Saisir un entier :
255
Conversion en octal : 377
Conversion en binaire : 11111111
Conversion en Hex : ff
Saisir un entier :
|
```

```
public class Client {

    public static void main(String[] args) {
        Saisi saisi=new Saisi();
        ConvBin convB=new ConvBin();
        ConvHex convH=new ConvHex();
        ConvOctale convO=new ConvOctale();

        saisi.add(convO);
        saisi.add(convB);
```

```

        saisi.add(convH);
        Scanner sc=new Scanner(System.in);

        while(true){
            System.out.println("Saisir un entier :");
            saisi.setNombre(sc.nextInt());
        }
    }
}

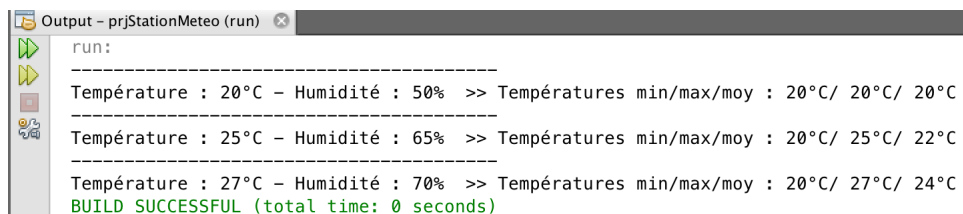
```

Exercice 2 :

Une station météo maintient une liste des données météorologiques (températures et les taux d'humidité). Ces données seront exploitées par 3 affichages différents :

- Affichage de la température et du taux d'humidité
- Affichage de la température minimum, maximum et de la moyenne

Chaque ajout d'une nouvelle données provoquera la mise à jour automatique des affichages.



```

package prjstationmeteo;

public class Client {

    public static void main(String[] args) {
        StationMeteo sm=new StationMeteo();
        AffichageCondition ac=new AffichageCondition(sm);
        AffichageStatistique as=new AffichageStatistique(sm);

        sm.addObservateur(ac);
        sm.addObservateur(as);

        sm.ajouterDonnees(20, 50);
        sm.ajouterDonnees(25, 65);
        sm.ajouterDonnees(27, 70);
    }
}

```

TP4 : Les composants GUI

Durée

3 séances

Objectifs

- Utilisation des composants `TextField`, `TextArea`, `Label`, `Button`, `TextArea` et `ScrollPane`
- Gestion des événements avec les classes internes et les classes anonymes
- Gestionnaires d'agencement (`FlowLayout`, `BorderLayout` et `BoxLayout`)
- Composants de choix
- Conception de menu

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice!!!

Package : `be.isims.ihm.tp1.ex#`

Exercice 1 : Compte bancaire

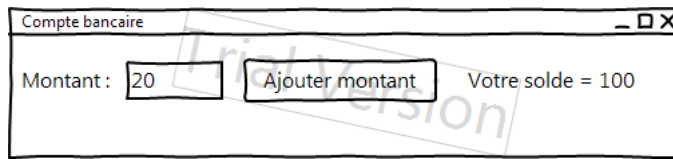
Réalisez une application **REUTILISANT** la classe `CompteBancaire` (classe métier) de l'exercice 2 du TP 2 (sans la changer!) et lui ajouter une interface graphique.

a) Conception de l'interface

L'interface sera constituée d'un composant `Label` pour afficher, à gauche du champ de texte, le texte "Montant :". Un second `Label` sera utilisé pour afficher le solde du compte.

Un champ de texte (`TextField`) permettra à l'utilisateur d'introduire le montant qu'il souhaite déposer sur son compte. Le bouton "Ajouter montant" (`Button`) sera utilisé pour ajouter un montant au solde. Ce dernier sera ajouté à l'aide du bouton (`Button`) "Ajouter montant".

Reproduisez l'interface graphique suivante :



La classe représentant la fenêtre sera nommée `CompteBancaireFrm` dont les dimensions seront 450 pixels pour sa largeur et 100 pixels pour sa hauteur. La classe principale `CompteBancaireTest` construira et affichera cette fenêtre.

b) Gestion de l'événement du bouton

Lorsque l'utilisateur clique sur le bouton, vous devez récupérer le montant et l'ajouter au solde du compte. L'application mettra à jour le texte du `JLabel` affichant le montant.

La gestion de cet événement se fera à l'aide d'une *classe interne* à la classe `CompteBancaireFrm`, nommée `AjouteMontantEcouteur`.

c) Affichage de l'historique du solde

Ajoutez une zone de texte (`JTextArea`) pour afficher les valeurs successives du solde. Équipez cette zone d'une barre de défilement (`JScrollPane`) afin de visualiser l'ensemble des valeurs.

Exercice 2 : Self banking

Réalisez une interface graphique permettant à l'utilisateur de réaliser des transactions sur ses deux comptes (courant et épargne).

Veillez à concevoir une application extensible!!!

a) Conception de l'interface

Reproduisez l'interface graphique suivante :

Pour réaliser cette interface vous allez avoir besoin du gestionnaire d'agence-ment `BoxLayout`. Le champ de texte permet à l'utilisateur de saisir le montant de la transaction. Le compte est sélectionné à l'aide de la liste déroulante (`JComboBox`). Le type de transaction (Dépôt ou Retrait) est choisi via les boutons radio (`JRadioButton`).

Le bouton "Valider" permet d'exécuter la transaction. Tandis que le bouton "Calculer les intérêts" permet comme son nom l'indique de calculer les intérêts du compte d'épargne.

Une zone de texte affiche l'historique des transactions.

Info utile pour le `JComboBox` : Vous devez savoir que lorsque un objet de type `JComboBox` affiche les éléments ajoutés, celui-ci appelle la méthode `toString()` des objets ajoutés.

Attention, les comptes créés doivent s'ajouter automatiquement dans la liste déroulante.

b) Gestion des événements et description de la logique métier

Utilisez des *classes anonymes* pour gérer les événements.

Un compte est caractérisé par son solde et sa dénomination. Les actions que l'on souhaite faire sur un compte sont ; verser un montant et retirer de l'argent.

La classe `CompteEpargne` est un compte qui possède un taux d'intérêt. L'opération `calculIntérêts` permet de mettre à jour le solde en tenant compte des intérêts. La méthode `toString()` à de cette classe retourne une chaîne de caractères de type : "<Compte Epargne> : solde = x Euros".

La classe `CompteCourant` est un compte pour lequel chaque opération de retrait est payante et vaut 5% du montant de l'opération. La méthode `toString()` de cette classe retourne une chaîne de caractères de type : « <Compte courant> :

solde = x Euros »

La classe `Banque` est caractérisée par un tableau (ou `ArrayList`) `lesComptes` destiné à contenir tous les comptes créés. Cette classe contient les méthodes suivantes :

- `addCompte` : permet d'ajouter le compte "XY" au tableau `lesComptes`
- `getCompte` : permet de retourner un compte à partir d'un index.
- `collectionComptes` : permet de retourner tous les comptes.

La classe `BanqueFrm` permet de :

- Créer un objet `Banque`.
- Créer un compte courant et un compte épargne, en les ajoutant au tableau `lesComptes`.
- Faire des opérations de retrait et de dépôt sur les comptes créés.

c) Ajoutez un nouveau type de compte à votre application

La classe `ComptePayant` est un compte pour lequel chaque opération de versement ou de retrait est payante et vaut 5% du montant de l'opération. La méthode `toString()` de cette classe retourne une chaîne de caractères de type : "<Compte payant> : solde = x Euros".

Exercice 3 : Calculatrice

Réalisez une calculatrice élémentaire permettant de réaliser les opérations de base : +, -, * et /

a) Réalisation de l'interface

Pour reproduire cette interface vous allez devoir utiliser les gestionnaires d'agencement : `BorderLayout` et `GridLayout`.

`String getActionCommand()` :

Renvoie la chaîne de commande associée à cet événement action. Si l'événement a été déclenché par un bouton, la chaîne de commande contient le libellé du bouton, à moins d'un changement intervenu par le biais de la méthode `setActionCommand`.

TP4 : Les composants GUI (suite)

Objectifs

- Gestion des événements
- Utilisation des composants graphiques
- Mise en œuvre du pattern MVC

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice !

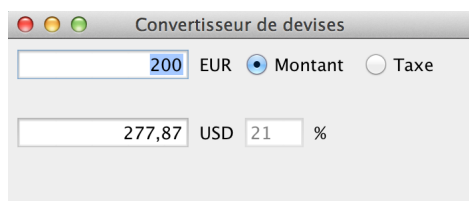
Package : `be.isims.ihm.tp2.ex#`

Exercice 1 : Convertisseur EUR-USD avec calcul de taxe

Réalisez une application permettant la conversion de devises et le calcul d'un montant avec taxe comprise.

Vous devez construire cette interface graphique sans utiliser le "designer GUI" de Netbeans.

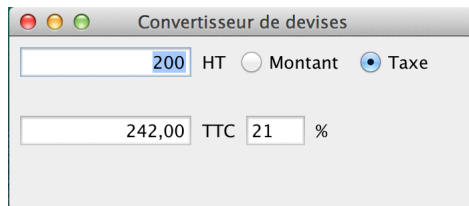
a) Conversion



L'application convertit les EUR(Euros) en USD(dollars américains) et vice-versa. Dès que l'utilisateur modifie la valeur dans un des deux champs textes, l'autre se met automatiquement à jour.

Le mode conversion est actif par défaut ou lorsque l'utilisateur sélectionne la case "Montant". Dans ce mode, le champ texte "21 %" est désactivé.

b) Calcul de taxe

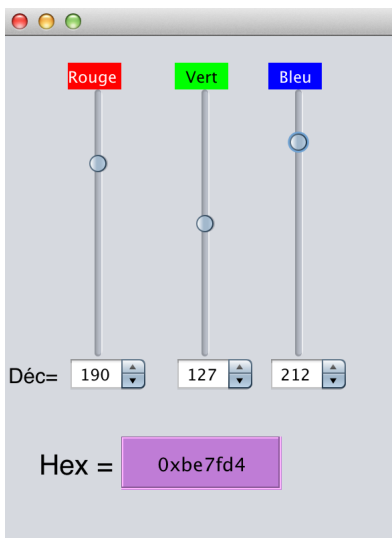


L'application bascule dans ce mode lorsque l'utilisateur sélectionne la case "Taxe". Au basculement, le montant TTC est calculé et affiché dans le champ "TTC". De plus, le champ "Taxe" devient alors disponible.

Dès que l'utilisateur modifie le taux, les calculs s'effectuent automatiquement entre le montant HT et le montant TTC.

Exercice 2 : Convertisseur et générateur de couleurs

Réaliser un programme permettant de régler les trois couleurs fondamentales (Rouge, Vert, Bleu) afin d'obtenir une nouvelle couleur.



Les valeurs des trois composantes RVB sont définies à l'aide :

- des trois curseurs de défilement (*JSlider*)
- des trois contrôles *Spinner*

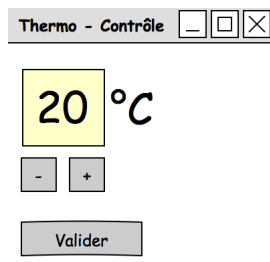
Un *JLabel* permet de visualiser la couleur générée et d'afficher sa valeur en hexadécimale.

Exercice 3 : Thermomètre

Réaliser une application simulant un thermomètre digital, sur laquelle l'utilisateur peut agir pour contrôler la température. Cette application doit être conçue suivant une architecture MVC.

a) Interface graphique

Une fenêtre permet de préciser la température soit à l'aide du champ de texte ou à partir des boutons (-) et (+). La valeur est enregistrée lorsque l'utilisateur clique sur le bouton "Valider".



Une autre fenêtre est destinée à l’affichage textuel de la température courante.



TP5 : Design pattern décorateur

Objectifs

Mise en œuvre du pattern décorateur.

Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Faites le diagramme de classes pour chaque exercice !

Package : `be.isims.ihm.tp3.ex#`

Exercice : Pizzas

Une pizzeria propose de composer sa pizza à partir de trois types de pizzas de base qui sont : la classique, la fourrée au fromage et la pâte fine.

À partir d'une pizza de base, un client peut ajouter dynamiquement des ingrédients au choix : boulettes de viande, thon, jambon, fromage, oignons, champignons et olives noires.

Chaque base de pizza et chaque ingrédient supplémentaire possède un prix fixe. Le coût de la pizza finale sera la somme des prix de ses composants.

Prévoyez les méthodes `afficheDescription` et `calculePrix` qui affichent pour chaque pizza sa description complète (ingrédients compris) et son prix final.

Programmez cette application en attribuant un prix et une description à chaque base et ingrédient puis testez-la en composant une pizza et en affichant sa description et son prix.

TP6 : Les fichiers

Objectifs

Manipulations des fichiers.

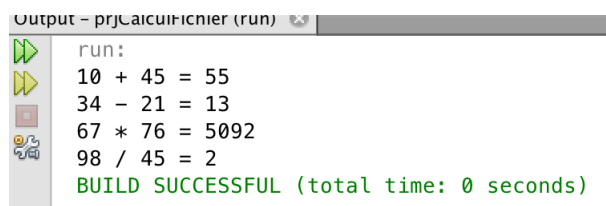
Remarques

Vos programmes doivent être conformes aux conventions d'écriture telles que définies sur le site suivant : <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Chaque classe doit avoir sa documentation interne et sa documentation d'interface de modules (javadoc).

Exercice 1

- Réaliser une classe **GenerateurFichier** avec une méthode **stockerNombres**, prenant en arguments un nom de fichier et un tableau d'entiers. Ajouter également un méthode **stockerCaracteres** avec en arguments, un nom de fichier et un tableau de caractères. Tester votre classe en générant trois fichiers *operande1.dat*, *operande2.dat* et *opérateurs.dat* avec respectivement un ensemble d'entiers, un autre ensemble d'entiers et une liste d'opérateurs +, -, * et /. Il faut le même nombre d'éléments entre chaque fichier
- On dispose des trois fichiers *operande1.dat*, *operande2.dat* et *opérateurs.dat*. Ecrire un programme (classe **CalculFichier**) lisant en parallèle ces trois fichiers et affichant dans la console chaque opération (entier opérateur entier) avec son résultat.



```
Output - prjCalculFichier (run)
run:
10 + 45 = 55
34 - 21 = 13
67 * 76 = 5092
98 / 45 = 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Exercice 2

Ecrire une classe **CopierSansBlanc** qui contient une méthode **copier**. Cette méthode à deux arguments, un fichier texte source et un fichier texte de destination. Le fichier texte de destination est une copie du fichier texte source à l'exception que tous les caractères blancs sont remplacés par un caractère '_'. Employer un buffer de 512 caractères pour vos traitements de lecture et d'écriture.

A partir d'un texte saisi dans votre éditeur favori, tester votre classe.

Rmq : La méthode statique **Character.isWhitespace** vous permet de détecter un caractère blanc.

Fichier source :

```

1  hsjdhjdhs djksjdk jkdj kdj hsjdh
2  dhsjhd n k k l m m mm m

```

Fichier de destination :

```

1  hsjdhjdhs_djksjdk_jkdj_kdj_hsjdh_dhsjhd_n_k_k_l_m_m_mm_m

```

Exercice 3

L'objectif de cet exercice est de réaliser un tableau d'entiers fonctionnant uniquement à travers un fichier, pour les opérations de consultation et d'écriture.

Créer une classe **FichierTab** avec les attributs **taille** et **fichier**, mise à jour respectivement par le constructeur pour définir la taille du tableau et par le fichier de stockage/consultation. Créer une méthode **ecrire** avec en argument, un index et une valeur entière, puis une méthode **lire** avec en argument un index. Lever une exception **java.lang.ArrayIndexOutOfBoundsException** en cas d'erreur d'index.

Rmq : Utiliser la méthode **seek** pour vous positionner directement à la bonne valeur (en n'oubliant pas qu'un entier est sur 4 octets).

```

run:
Indice 0 : 12
Indice 5 : 34
Indice 10 : 100
BUILD SUCCESSFUL (total time: 0 seconds)

```

```

public static void main(String[] args) {
    FichierTab ft=new FichierTab(20, "tabInteger");
    ft.ecrire(0, 12);
    ft.ecrire(5, 34);
    ft.ecrire(10, 100);

    System.out.println("Indice 0 : "+ft.lire(0));
    System.out.println("Indice 5 : "+ft.lire(5));
    System.out.println("Indice 10 : "+ft.lire(10));
}

```

Troisième partie

Réseaux et objets distribués

Quatrième partie

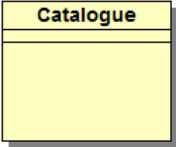
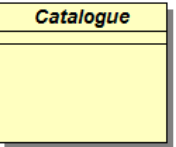
Annexe

Annexe A

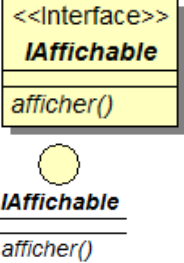

Correspondances UML - Java

La construction de ce tableau a été inspiré des livres de Hugues Bersini[1] et Pascal Roques[7]

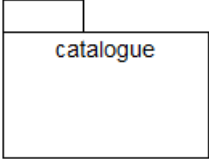
A.1 Classe

UML	Java
 Classe	<pre>public class Catalogue { ... }</pre>
 Classe abstraite	<pre>public abstract class Catalogue { ... }</pre>

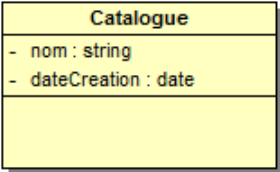
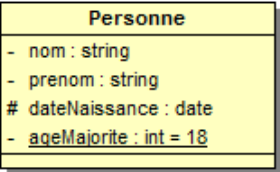
A.2 Interface

UML	Java
 IAffichable afficher()  IAffichable afficher()	<pre>public interface IAffichable { public void afficher(); }</pre>

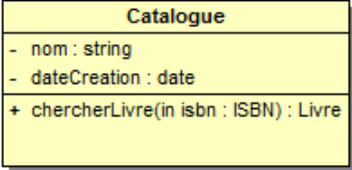
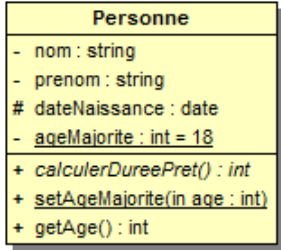
A.3 Package

UML	Java
	<pre>package catalogue; ...</pre>

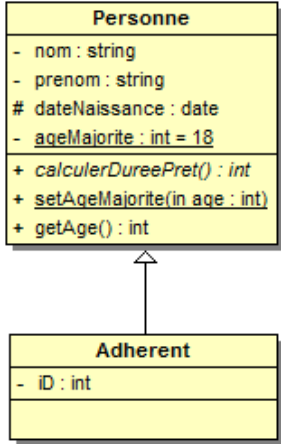
A.4 Attribut

UML	Java
 Classe	<pre>import java.util.Date; public class Catalogue { private String nom; private Date dateCreation; ... }</pre>
 Classe abstraite	<pre>public class Personne { private String nom; private String prenom; protected Date dateNaissance; private static int ageMajorite = 18; }</pre>

A.5 Méthode (opération)

UML	Java
 <pre> classDiagram class Catalogue { - nom : string - dateCreation : date + chercherLivres(in isbn : ISBN) : Livre } </pre>	<pre> public class Catalogue { private String nom; private Date dateCreation; public Livre ChercherLivres(ISBN isbn) { ... } ... } </pre>
 <pre> classDiagram class Personne { - nom : string - prenom : string # dateNaissance : date - ageMajorite : int = 18 + calculerDureePret() : int + setAgeMajorite(in age : int) + getAge() : int } </pre>	<pre> public class Personne { private String nom; private String prenom; protected Date dateNaissance; private static int ageMajorite = 18; public abstract int calculerDureePret(); public static void setAgeMajorite(int age) { ... } public int getAge() { ... } } </pre>

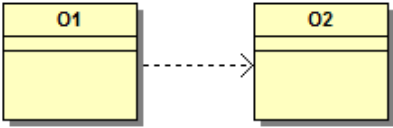
A.6 Héritage (généralisation)

UML	Java
 <pre> classDiagram class Personne { - nom : string - prenom : string # dateNaissance : date - ageMajorite : int = 18 + calculerDureePret() : int + setAgeMajorite(in age : int) + getAge() : int } class Adherent { - id : int } Personne < -- Adherent </pre>	<pre> public class Adherent extends Personne { private int id; } </pre>

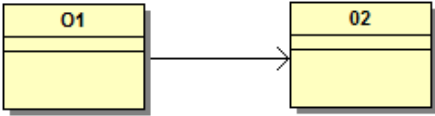
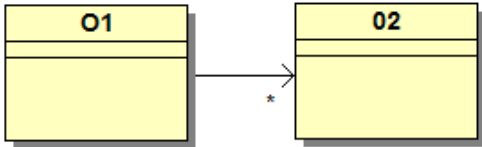
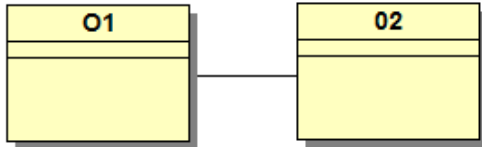
A.7 Réalisation

UML	Java
<pre> classDiagram class IEmpruntable { <<interface>> +emprunter() void +retourner() void } class IImprimable { <<interface>> +imprimer() void } class Livre { -titre : string -auteur : string -isbn : ISBN +emprunter() void +retourner() void +imprimer() void } IEmpruntable < .. Livre IImprimable < .. Livre </pre>	<pre> public class Livre implements IImprimable, IEmpruntable { private String titre; private String auteur; private ISBN isbn; public void imprimer(){ ... } public void emprunter(){ ... } public void retourner(){ ... } } </pre>

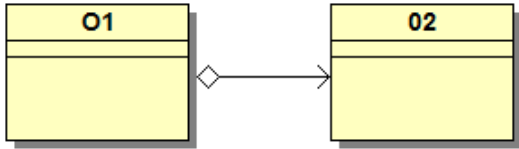
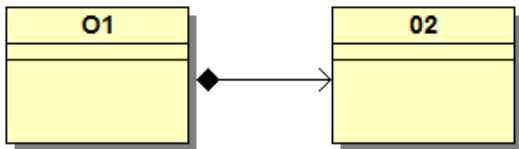
A.8 Dépendance

UML	Java
 <pre>classDiagram class 01 class 02 01 ..> 02</pre>	<pre>class 01 { private int unAttribut; private int unAutreAttribut ; public void jeTravaillePour01() { 02 lien02 = new 02() ; lien02.jeTravaillePour02() ; } // l'objet lien02 disparaît public int uneAutreMethode(int a){ return a; } } OU class 01 { void jeTravaillePour01(02 lien02) { lien02.jeTravaillePour02() ; } }</pre>

A.9 Association

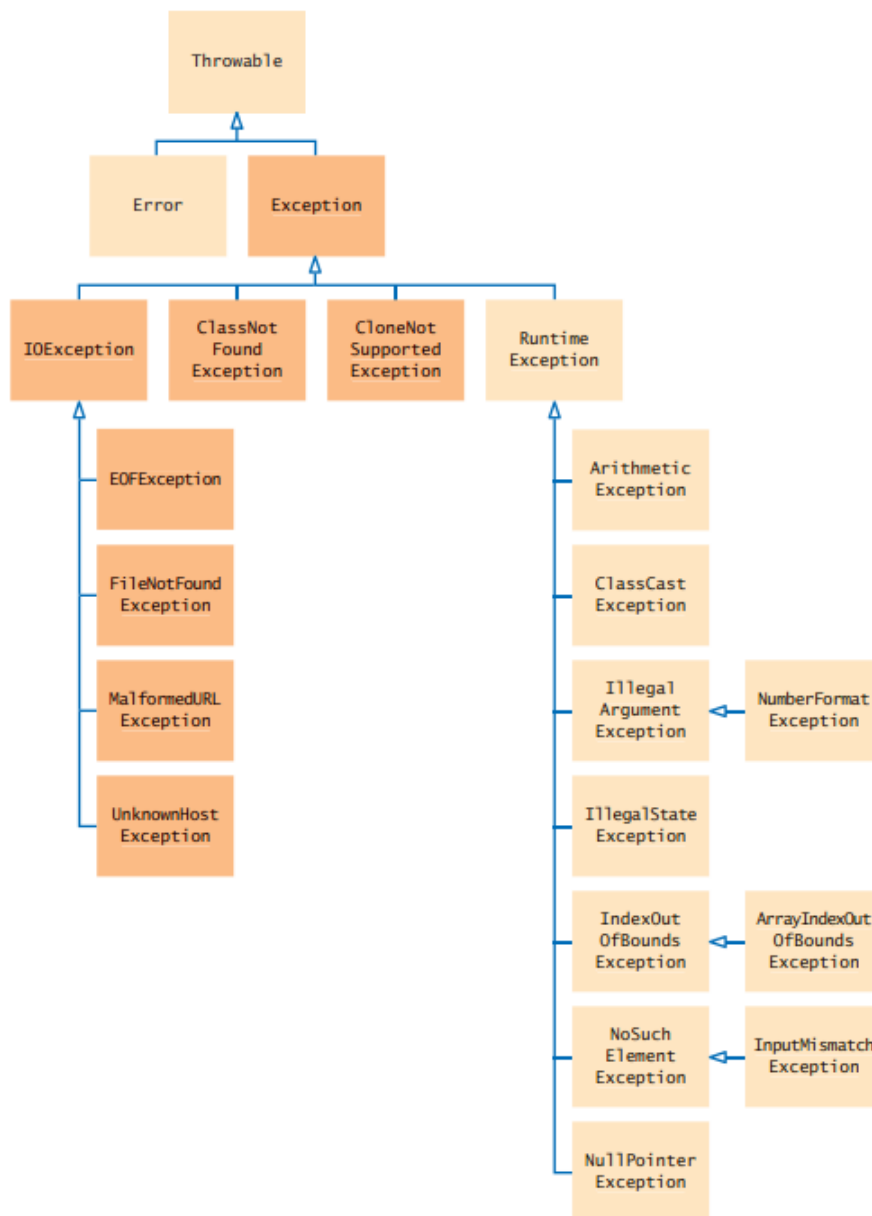
UML	Java
	<pre>public class 01 { private 02 lien02; ... }</pre>
	<pre>class 01 { 02[] lesLiens02 ; // ou ArrayList<02> lesLiens02 ; }</pre>
	<pre>public class 01 { private 02 lien02; ... } public class 02 { private 01 lien01; ... }</pre>

A.10 Agrégation et composition

UML	Java
 <p>Agrégation "contient ou "est composé de"</p>	<pre>class 01 { private 02 lien02; public 01(02 lien02) { this.lien02 = lien02; } }</pre>
	<pre>class 01 { private 02 lien02; public 01() { lien02 = new 02(); } }</pre> <p>OU</p> <pre>public class 01 { private class 02 { // classe interne (imbriquée) } ... }</pre>

Annexe B

La hiérarchie des classes d'exception



Annexe C

Résumé de la gestion des événements

<i>Interface</i>	<i>Méthodes</i>	<i>Paramètres/ accesseurs</i>	<i>Événements générés par</i>
ActionListener	actionPerformed	ActionEvent • getActionCommand • getModifiers	AbstractButton JComboBox JTextField Timer
AdjustmentListener	adjustmentValueChanged	AdjustmentEvent • getAdjustable • getAdjustmentType • getValue	JScrollbar
ItemListener	itemStateChanged	ItemEvent • getItem • getItemSelectable • getStateChange	AbstractButton JComboBox
FocusListener	focusGained focusLost	FocusEvent • isTemporary	Component
KeyListener	keyPressed keyReleased keyTyped	KeyEvent • getKeyChar • getKeyCode • getKeyModifiersText • getKeyText • isActionKey	Component
MouseListener	mousePressed mouseReleased mouseEntered mouseExited mouseClicked	MouseEvent • getClickCount • getX • getY • getPoint • translatePoint	Component
MouseMotionListener	mouseDragged mouseMoved	MouseEvent	Component
MouseWheelListener	mouseWheelMoved	MouseWheelEvent • getWheelRotation	Component

<i>Interface</i>	<i>Méthodes</i>	<i>Paramètres/ accesseurs</i>	<i>Événements générés par</i>
WindowListener	windowClosing windowOpened windowIconified windowDeiconified windowClosed windowActivated windowDeactivated	WindowEvent • getWindow	Window
WindowFocusListener	windowGainedFocus windowLostFocus	WindowEvent • getOppositeWindow	Window
WindowStateListener	windowStateChanged	WindowEvent • getOldState • getNewState	Window

Annexe D

Ressources

D.1 Liens utiles

<http://www.oracle.com/us/technologies/java/index.html>
<http://javablogs.com/Welcome.action>
<http://blog.alexis-hassler.com/>
<http://www.alpesjug.fr/>
<http://www.siteduzero.com/>
<http://java.developpez.com/>
<http://www.eclipse.org/>
<http://uml.free.fr/>

D.2 Littératures utiles

Daniel Pourbaix. *cours TR1T11*. HeH-ISIMs, 2011
Bertrand Meyer. *Conception et programmation orientées objet*. Eyrolles, 2008
Kathy Sierra and Bert Bates. *Head First Java*. O'Reilly, 2008
Magazine : Programmez !

Bibliographie

- [1] Hugues Bersini. La programmation orientée objet. Groupe Eyrolles, 2009.
- [2] Paul Deitel. Java How to Program : Early Objects Version, 8th ed. Prentice Hall, 2009.
- [3] Jason De Oliveira et Fathi Bellahcene. Un code de qualité est un code s.o.l.i.d.(e)! Programmez !, pages 61–62, Septembre 2010.
- [4] Cay Horstmann. Big Java : Compatible with Java 5, 6, and 7, 4th ed. John Wiley and Sons Ltd, 2010.
- [5] Cay Horstmann and Gary Cornell. Au coeur de Java, volume 1 : Notions fondamentales (Java SE 6), 8e édition. Pearson Education, 2008.
- [6] John Lewis and William Loftus. Java Software Solutions : Foundations of Program Design, 4th ed. Addison Wesley, 2004.
- [7] Pascal Roques. UML par la pratique, 5 édition. Eyrolles, 2006.