IUT d’Orsay - BUT Informatique - Jean-Claude MARTIN – [jean-claude.martin@universite-paris-saclay.fr](mailto:jean-claude.martin@universite-paris-saclay.fr)

# S2 R2.01 - Développement Orienté Objet

# EX 6 : Interfaces

Objectifs

* Comprendre l’utilité d’une interface
* Programmer des interfaces
* Liens entre interfaces et classes abstraites

# AbstractAction

*Dans cet exercice, nous allons apprendre à comprendre la description d’une classe abstraite fournie dans le JDK et qui implémente plusieurs interfaces.*

Voici un extrait de la javadoc:

public abstract class AbstractAction

extends Object

implements Action, Cloneable, Serializable

This class provides default implementations for the JFC Action interface. Standard behaviors like the get and set methods for Action object properties (icon, text, and enabled) are defined here. The developer needs only subclass this abstract class and define the actionPerformed method.

Vous n’avez pas à écrire de programme qui utilise cette classe.

Vous devez seulement répondre aux questions ci-dessous :

1. Est-ce que AbstractAction est une classe concrète ? … non............
2. Est-ce que AbstractAction est une classe abstraite ? .... oui...........
3. Est-ce que AbstractAction est une interface ? ....non...........

Qu’est-ce que cela signifie pour le développeur qui veut utiliser AbstractAction :

1. Qu’est-ce qu’il ne peut pas faire ? ....... on peut pas créer un object directment......................................
2. Qu’est-ce qu’il doit faire pour utiliser cette classe abstraite ?

... .on peut créer une classe concrète.qui doit vraiment implementer la classe abstraite .................................................................................................

.........................................................................................................

# Héritage, classes abstraites, interfaces : Salariés

*Dans cet exercice, nous allons voir une des utilités d’une interface dans le cas où nous avons déjà une hiérarchie de classes (abstraites et concrètes). L’interface permet de regrouper des classes à différents endroits de la hiérarchie et qui hérite déjà d’une classe (donc on ne pourrait pas les faire hériter d’une autre classe abstraite et on va donc plutôt les regrouper via une interface qu’elles implémenteront).*

Le directeur d'une entreprise souhaite gérer les salaires et primes de ses employés au moyen d'un programme Java.

QUESTION 1). Vous devez écrire un programme Java permettant de gérer ce qui est décrit ci-dessous.

Vous commencerez par faire un schéma UML.

Vous déciderez vous même si vous devez définir des classes concrètes, des classes abstraites et/ou des interfaces. Vous devrez aussi choisir les relations d’héritage et d’implémentation appropriées.

Un employé est caractérisé par

son nom, son prénom, son âge, son année d’entrée dans l'entreprise ;

une méthode calculerSalaire (ce calcul dépend du type de l'employé) ;

une méthode getNom retournant une chaîne de caractères obtenue en concaténant la chaîne de caractères "L'employé " avec le prénom et le nom ;

si cela est faisable, un constructeur permettant d’initialiser les attributs.

Le calcul du salaire mensuel dépend du type de l'employé. On distingue les types d'employés suivants :

Ceux affectés à la Production (les techniciens). Leur salaire vaut le nombre d'unités produites par mois par cet employé multiplié par 10.

Ceux affectés à la Manutention (les manutentionaires). Leur salaire vaut leur nombre d'heures de travail mensuel de cet employé multiplié par 15 Euros.

Ceux affectés à la Vente directe (les vendeurs). Leur salaire mensuel est 20 % du chiffre d'affaires qu’ils réalisent mensuellement (eux à titre individuel : ce sont les ventes que fait ce vendeur dans le mois), plus 200 Euros.

Ceux affectés à la Représentation (les représentants). Leur salaire mensuel est de 30 % du chiffre d'affaires qu'ils réalisent mensuellement (idem), plus 500 Euros.

Chaque sous classe est dotée de constructeur prenant en argument l'ensemble des attributs nécessaires.

N'hésitez pas à introduire des classes intermédiaires pour éviter au maximum les redondances d'attributs et de méthodes dans les sous-classes.

Ecrire un main() dans une classe Salaires pour tester ces classes et ces méthodes.

b)

Certains employés des secteurs production et manutention sont appelés à fabriquer et manipuler des produits dangereux et ont une prime mensuelle. Ajouter également à votre programme une spécification pour ces employés à risque permettant de leur associer une prime mensuelle fixe (donc une constante définie) de 250 Euros. On veut pouvoir bénéficier du polymorphisme, par exemple en gérant de manière homogène des **employés à risque en calculant la somme des salaires de tous les employés à risques stockés dans un tableau d’employés à risque**.

Donner un exemple dans le main().

On vous donne la classe suivante :

**class** Personnel {

**private** Employe[] staff;

**private** **int** nbreEmploye;

**private** **final** **static** **int** ***MAXEMPLOYE*** = 200;

**public** Personnel() {

staff = **new** Employe[***MAXEMPLOYE***];

nbreEmploye = 0;

}

**public** **void** ajouterEmploye(Employe e) {

**if** (nbreEmploye <= ***MAXEMPLOYE***) {

++nbreEmploye;

staff[nbreEmploye - 1] = e;

} **else** {

System.***out***.println("Pas plus de " + ***MAXEMPLOYE*** + " employés");

}

}

**public** **double** salaireMoyen() {

**double** somme = 0.0;

**for** (**int** i = 0; i < nbreEmploye; i++) {

somme += staff[i].calculerSalaire();

}

**return** somme / nbreEmploye;

}

**public** **void** afficherSalaires() {

**for** (**int** i = 0; i < nbreEmploye; i++) {

System.***out***.println(staff[i].getNom() + " gagne "+ staff[i].calculerSalaire());

}

}

QUESTION 2) parmi les méthodes qui sont définies et celles qui sont appelées dans cette classe Personnel, quelle est (ou quelles sont) la (ou les) méthodes polymorphes ?

# Gestion d’albums photos

*Dans la question 1 de cet exercice, nous allons voir comment implémenter une interface existante fournie avec le JDK (Comparable) qui permettra ensuite d’appeler une méthode de tri.*

*Dans la question 2 de cet exercice, nous allons voir une des utilités d’une interface : découper le développement informatique en deux phases : une phase de spécification dans laquelle on définit l’interface, puis une phase d’implémentation dans laquelle on définit une classe concrète qui implémente l’interface.*

Vous devez écrire un programme qui permet de gérer des photos de vacances.

Une photo est caractérisée par le nom du pays dans lequel cette photo a été prise, une année de prise de vue, un commentaire, un nom de fichier et la taille du fichier.

1) Photo

Définissez la classe Photo dans le package photos.modeles.

Vous définirez un constructeur qui permettra d’initialiser les attributs. L’attribut taille du fichier sera initialisé à l’aide de la classe File dont voici un exemple d’utilisation dont vous pouvez vous inspirer :

import java.io.File;

public class FileSizeExample {

public static void main(String[] args) {

File file =new File("image.jpg");

if(file.exists()){

// Récupérer la taille du fichier

double bytes = file.length();

}

Déplacer des fichiers .jpg par glisser coller depuis une fenetre explorateur Windows vers votre projet Eclipse (les fichiers .jpg doivent alors apparaître dans Eclipse à la racine de votre projet EX-06).

On voudra pouvoir comparer deux photos sur leur taille de fichier (regarder la partie du polycopié 6 sur les interfaces : interface Comparable).

Ecrire une méthode main dans la classe Photo qui crée 2 photos et qui les comparent.

2) *Pour gérer des albums photo, nous allons commencer par un niveau abstrait en définissant une interface.*

*Cela vous permettra de réfléchir aux méthodes que vous souhaiteriez pouvoir appeler sur un ensemble de photos, et ce quelle que soit son implémentation (qui pourrait ensuite être réalisée par un autre développeur que vous).*

*Vous pourriez aussi commencer à écrire des programmes qui utilisent cette interface par exemple avec des variables de type “EnsemblePhotos” sur lesquelles vous appelez ces méthodes.*

*Puis dans un second temps nous programmerons une classe concrète qui implémentera cette interface.*

*(Cela pourrait être fait par 2 personnes différentes : une qui définit l’interface, une autre personnes définissant la classe concrète)*

a) Définissez tout d’abord une **interface EnsemblePhotos**  
 qui permette de définir les spécifications : on voudra pouvoir :

ajouter une nouvelle photo dans cet ensemble,

trier les photos de l’ensemble selon la taille du fichier,

rechercher dans l’ensemble de photos un mot-clé dans les commentaires et retourner un arrayList de références des photos qui ont ce mot-clé dans leur commentaire.

b)

Ecrire une classe AlbumPhoto dans le package photos.modeles.

Celle classe devra permettre de représenter un ensemble de photos.

Proposez ensuite une implémentation de cette spécification en définissant la classe **AlbumPhoto**.

Les photos seront stockées dans un ArrayList.

Pour information, l’instruction Collections.sort(arrayList) permet de trier l’arrayList passé en paramètre.

(Arrays.sort() ne trie que des tableaux, pas des ArrayLists).   
Il faut bien sûr que les éléments stockés dans l’array list soient comparables…

Note :

Il est possible de restreindre les types comparés avec Comparable<Type>.

# Banque : interface Compte

*Dans cet exercice, nous allons voir comment améliorer un exercice que nous avons déjà fait et dans lequel nous n’avions pas encore à notre disposition cet outil que sont les interfaces.*

Le projet Banque vu à la feuille précédente ne comporte pas de niveau complètement   
abstrait d’un compte et des méthodes qu’on peut appeler dessus.

Dupliquer votre projet Banque.

Définissez une interface Comptable

avec des spécifications de méthodes : débiter, crediter, getSolde.

Implémentez ensuite cette interface avec la classe abstraite Compte   
pour gérer les comptes courants et les comptes d’épargne.

Normalement, il faudrait procéder dans l’ordre inverse :

définir l’interface puis définir la classe abstraite, puis définir les classes concrètes.

# Mediane (paramètre d’une méthode de type interface)

*Dans cet exercice, nous voyons comment les interfaces permettent le polymorphisme des paramètres d’une méthode. Le paramètre mm de la méthode mediane peut prendre plusieurs formes à l’exécution : cela pourra être une référence vers une instance de la classe* MinMaxPaire ou une *référence vers une instance de la classe* MinMaxTriplet.

Ecrire l’interface MinMax et les classes concrètes nécessaires pour pouvoir exécuter le programme ci-dessous.

**class** Mediane {

// Polymorphisme au niveau du paramètre qui est de type "interface"

**int** mediane(MinMax mm) {

**return**((mm.minimum()+mm.maximum())/2) ;

}

**public** **static** **void** main (String args []) {

Mediane med = **new** Mediane ();

MinMaxPaire mmPaire = **new** MinMaxPaire ();

mmPaire.setV1 (10);

mmPaire.setV2 (20);

System.***out***.println ("Valeur mediane pour une paire : " + med.mediane (mmPaire));

MinMaxTriplet mmTriplet = **new** MinMaxTriplet();

mmTriplet.setV1 (10);

mmTriplet.setV2 (20);

mmTriplet.setV3 (30);

System.***out***.println ("Valeur mediane pour un triplet : " + med.mediane (mmTriplet));

}

}

Résultat de l’exécution :

Valeur mediane pour une paire : 15

Valeur mediane pour un triplet : 20