

Lycée  
DIDEROT

polyvalent

Département IRIS

61, rue David d'Angers

75019 PARIS

<http://www.diderot.org>



# Java - Activités pratiques

## Travaux pratiques sur le langage JAVA

Auteur	Date - Version	Nom du fichier
Génaël VALET	version 3.2 - Juin 2007	activites-pratiques-java.docx

Ce document est un recueil d'activités pratiques destiné à l'apprentissage des bases du langage JAVA : Syntaxe, boucles, programmation orientée objet, ....

Le support de cours « Cours n°1 – Notions fondamentales » est indispensable

# A. Sommaire

- A. SOMMAIRE ..... 2
- B. OBJECTIFS..... 3
- C. ACTIVITES ..... 3
  - C.1. Compiler et exécuter un programme Java..... 3
    - a. Objectifs .....3
    - b. Création d'un projet, d'un package et d'une classe.....3
    - c. Exécution de la classe.....6
    - d. Exécution du projet avec passage d'arguments .....7
  - C.2. Construire un exécutable Java (Jar)..... 7
  - C.3. Tri de tableau ..... 8
  - C.4. Héritage et classes abstraites : *Forme, Ellipse, Cercle*..... 9
    - a. Objectifs .....9
    - b. Création des classes *Forme, Ellipse et Cercle* .....9
    - c. Création de la classe *Rectangle et Carre* .....10
    - d. Pour aller plus loin .....10
  - C.5. Agrégation : *Gestion d'albums musicaux*..... 11
    - a. Objectifs .....11
    - b. Classe « *Duration* » .....11
    - c. Classe « *Piste* » .....12
    - d. Classe « *Album* » .....12
    - e. Classe « *TestAlbums* » .....13
  - C.6. Agrégation et Collections : *Gestion de compte bancaire*..... 14
    - a. Objectifs .....14
    - b. Classe « *NumeroCompte* » .....14
    - c. Classe « *Operation* » .....15
    - d. Classe *CompteBancaire*.....15
    - e. Gestionnaire de comptes bancaires : Classe *ComptesManager* .....15
    - f. Test de l'ensemble.....16

## B. Objectifs

Ce document vous permettra d'être confronté à différents problèmes de programmation orientée objet en Java et de les résoudre en utilisant les concepts purement objets

## C. Activités

### C.1. Compiler et exécuter un programme Java

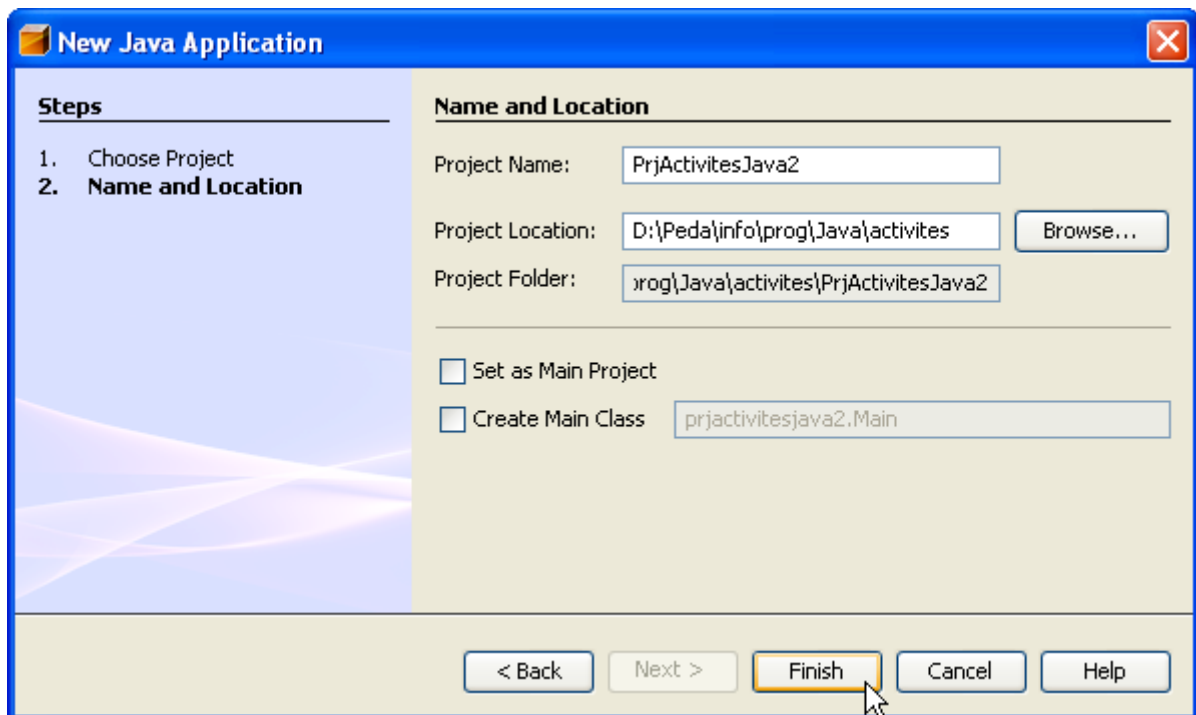
#### a. Objectifs

Compiler et exécuter un programme JAVA à partir du fichier .java fourni en utilisant l'IDE Netbeans.

#### b. Création d'un projet, d'un package et d'une classe

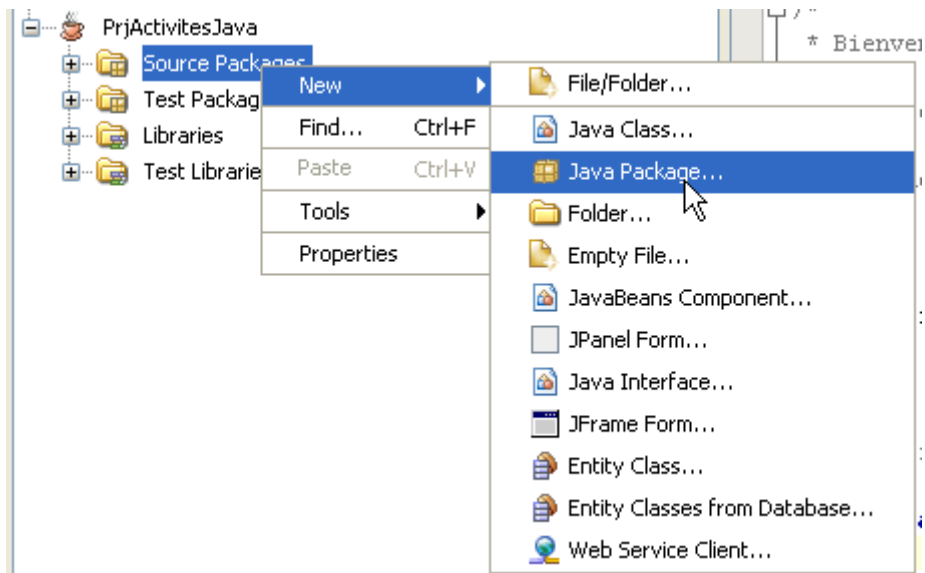
☞ Démarrez l'application « Netbeans » et créez un nouveau projet appelé : « **PrjActivitesJava** »

**Menu File > New Project > General > Java Application**

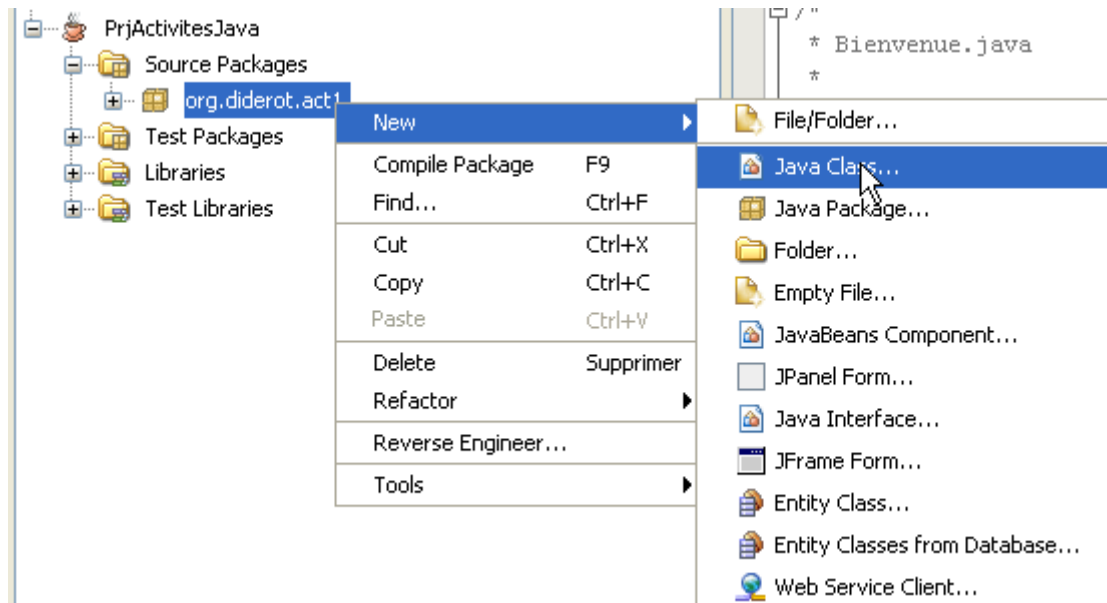


☞ Ensuite, créez un nouveau package intitulé : `org.diderot.activite1`

Voir la figure ci dessous



☞ Sous l'onglet « Projects », Cliquez à droite sur « Source Packages » > New > Java Class > Donnez le nom « Bienvenue » à votre nouvelle classe




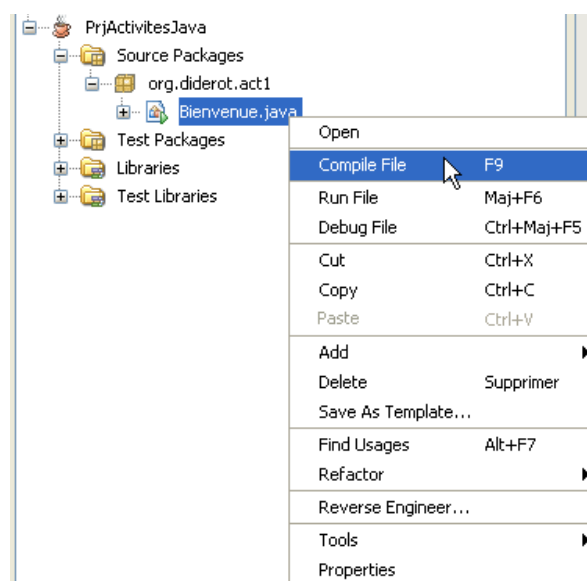
☞ Double cliquez sur la nouvelle classe qui doit s'ouvrir dans la fenêtre centrale


☞ Copiez ensuite le code suivant dans le corps de classe

D:\Peda\info\prog\Java\activites\PrjActivitesJava\src\org\diderot\act1\Bienvenue.java

```
2  * Bienvenue.java
3  *
4  * Created on 6 juin 2007, 11:00
5  *
6  * Affiche un message de bienvenue ou les arguments passés en ligne de commande
7  *
8  */
9
10 package org.diderot.act1;
11
12 /**
13  *
14  * @author G.VALET
15  */
16 public class Bienvenue {
17
18     public static void main ( String [] args) {
19
20         String ret = System.getProperty("line.separator");
21
22         if (args.length>0) {
23             System.out.println("Les arguments passés en paramètres sont :" + ret );
24
25             for (int i=0;i<args.length;i++)
26                 System.out.println(args[i]+ret);
27
28         } else
29             System.out.println(ret + "Bienvenue dans l&apos;univers Java" + ret);
30     }
31 }
32
```

 Compilez ensuite le programme en cliquant sur le bouton de droite sur le nom de la classe dans l'onglet « Project »




 Vérifiez que vous n'obtenez aucune erreur de compilation

```
Output - PrjActivitesJava (compile-single)  
init:  
deps-jar:  
compile-single:  
BUILD SUCCESSFUL (total time: 0 seconds)|
```

### c. Exécution de la classe

La classe possédant une méthode « main », il est donc possible de l'exécuter. Il existe 2 méthodes pour exécuter une classe :

- Sans arguments
- Avec des arguments passés en paramètres qui seront stockés dans le tableau de *String* nommé « args »

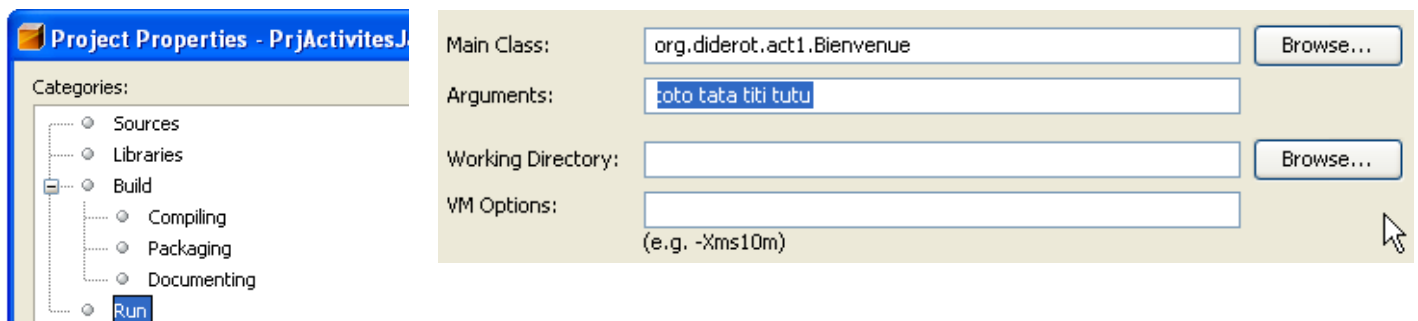
 Pour une exécution sans arguments, utilisez la touche Maj+F6 ou cliquez à droite sur la classe puis « Run File »

La fenêtre « Output » en bas de l'écran renvoie toutes les informations d'exécution, y compris les différents appels à « System.out.println »

```
Output - PrjActivitesJava (run-single)  
init:  
deps-jar:  
compile-single:  
run-single:  
  
Bienvenue dans l'univers Java  
  
BUILD SUCCESSFUL (total time: 0 seconds)|
```

#### d. Exécution du projet avec passage d'arguments

✎ Pour une exécution avec passage d'arguments, il suffit d'afficher les propriétés du projet : Clic droit sur le projet dans l'onglet « Project » et clic sur « Propriétés ». Clic sur « Run » et saisie de la classe principale « Main Class » et des arguments



✎ Ensuite, pour lancer l'exécution du projet, clic droit sur le projet puis « Run Project »

A l'exécution, Netbeans génère la sortie standard « Output - PrjActivitesJava »

```
Output - PrjActivitesJava (run)
init:
deps-jar:
compile:
run:
Les arguments passés en paramètres sont :

toto

tata

titi

tutu

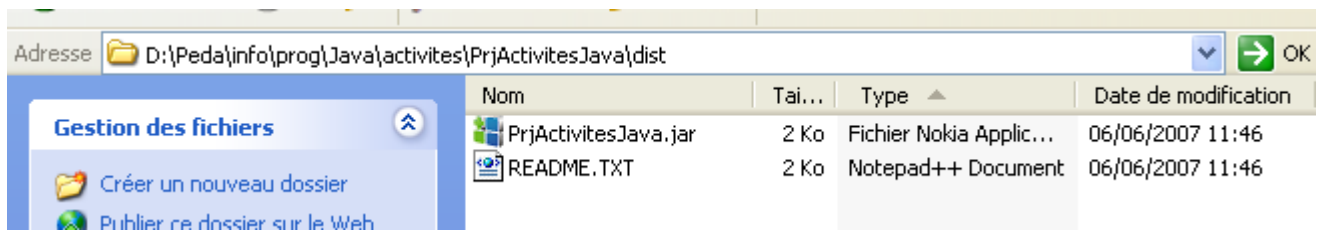
BUILD SUCCESSFUL (total time: 0 seconds)|
```

## C.2. Construire un exécutable Java (Jar)

L'idéal lorsqu'un programme est fonctionnel, est de pouvoir en faire un programme autonome pouvant s'exécuter à l'extérieur de l'IDE. Pour cela, Java utilise des formats de fichiers jar (Java Archive). Ces fichiers contiennent les classes (.class) compilées sans les fichiers sources (Bien qu'il soit possible de les inclure).

✎ Pour créer un fichier .jar exécutable, il déterminer les fichiers qui doivent être inclus dans l'archive. Ce processus est automatique pour Netbeans. Il vous suffit de cliquer à droite sur le nom du projet et de choisir « Build project »

Un fichier est alors créé dans le sous répertoire « dists » du projet



✎ Pour exécuter cette nouvelle application, ouvrez une invite de commande et déplacez-vous dans le répertoire contenant l'archive et tapez :

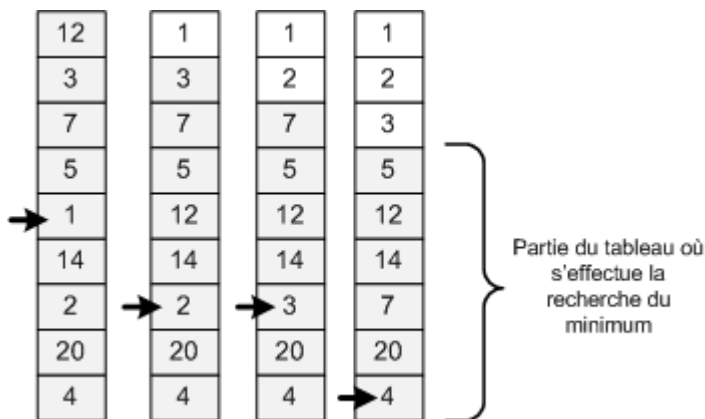
```
java -jar PrjActivitesJava.jar arg1 arg2 arg3
```

Le programme s'exécute et renvoie la sortie standard vers l'invite de commande.

### C.3. Tri de tableau

L'objectif de cette activité est la mise en œuvre d'un tri de tableau d'entiers selon la méthode du minimum. Le tableau d'entier sera déclaré en Java de la manière suivante :

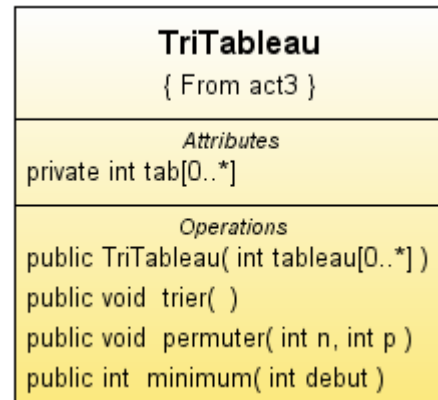
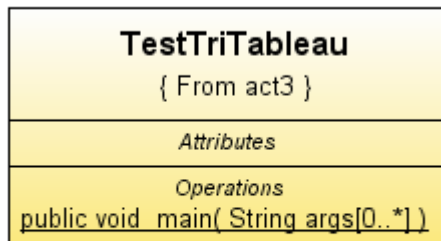
```
int [ ] tableau = { 12, 3, 7, 5, 1, 14, 2, 20, 4 } ;
```



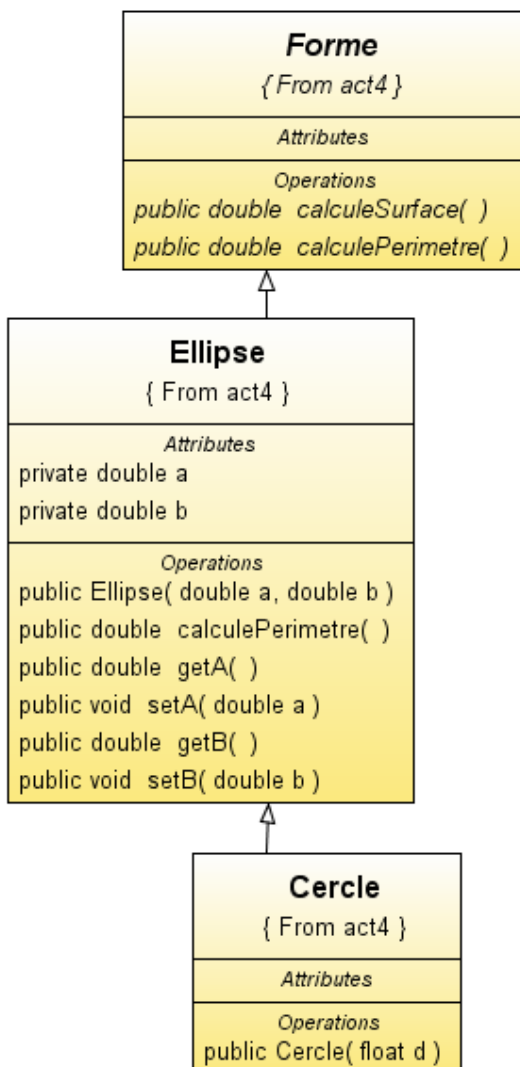
La recherche du nombre min s'effectue d'abord sur tout le tableau, puis sur ce même tableau diminué d'un élément. Si cette recherche est fructueuse, le programme effectue **la permutation avec l'élément pointé par la flèche**. L'opération est renouvelée jusqu'au dernier élément du tableau.

✎ Créez une classe nommée « TriTableau » et « TestTriTableau » permettant de résoudre de problème. Le programme doit afficher le tableau trié par ordre croissant. Le diagramme UML suivant vous indique les méthodes et champs des 2 classes à créer

La classe « TriTableau » contient différentes méthodes (operations) dont le rôle est décrit dans la documentation disponible à l'adresse suivante : <http://gita-server/java>



### C.4. Héritage et classes abstraites : Forme, Ellipse, Cercle



#### a. Objectifs

L'objectif de cette activité est de mettre en évidence l'utilisation de l'héritage et notamment des classes abstraites afin de définir un modèle de classe générique pouvant être réutilisé par les classes enfants.

Sur le diagramme ci-contre, on peut voir que la classe **Forme** est abstraite (En italique) car elle définit 2 méthodes abstraites :

- calculerSurface( )
- calculerPerimetre( )

Ces méthodes ne seront implémentées que dans la classe **Ellipse**. Cette classe définit 2 dimensions a et b représentant les dimensions de l'ellipse

La classe Cercle étant un cas particulier d'ellipse ( a=b), on réutilisera le constructeur de la superclasse **Ellipse** pour instancier les objets.

#### b. Création des classes Forme, Ellipse et Cercle

☞ Créez les classes *Forme*, *Ellipse* et *Cercle* en respectant le diagramme UML ci-dessus.

☞ Créez une classe de test appelée *TestFormes* afin de tester les différents calculs

### c. Création de la classe *Rectangle* et *Carre*

☞ Créez sur le même modèle les classes *Rectangle* et *Carre*.

### d. Pour aller plus loin

En examinant le code de *Rectangle* et *Ellipse*, on constate qu'il y a du code redondant pour la gestion des dimensions *a* et *b* de la forme. Comment prendre en considération le fait que les formes *Ellipse* et *Rectangle* sont des formes qui n'ont que 2 dimensions ?

☞ Réponse :

☞ Mettez en œuvre la solution

Pour vous aider, quelques rappels de mathématiques :

Aire de l'ellipse

$$A = \pi \cdot a \cdot b$$

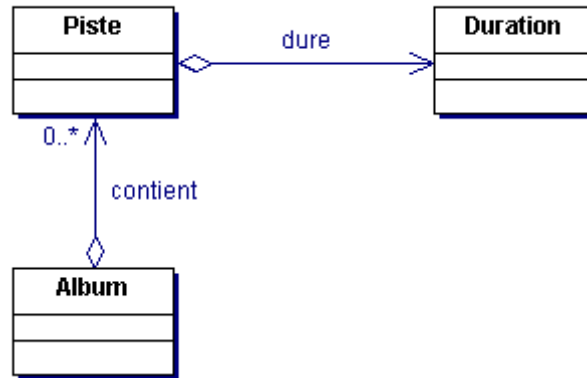
Périmètre de l'ellipse

$$P = 2 \cdot \pi \cdot \sqrt{\frac{1}{2} \cdot (a^2 + b^2)}$$

## C.5. Agrégation : Gestion d'albums musicaux

### a. Objectifs

Créer des classes permettant la gestion d'albums de musique. Les relations d'agrégation entre les différentes classes sont représentées dans le schéma ci-dessous :



On considère les points suivants :

- Chaque piste d'un album a une durée
- Chaque album contient de 0 à n pistes
- La classe Piste sera capable de renvoyer la durée du morceau
- La classe Album sera capable de calculer la durée totale de l'album

### b. Classe « Duration »

Duration
<i>Attributes</i>
private int hours private int minutes private int seconds
<i>Operations</i>
public Duration( ) public Duration( int theHours, int theMinutes, int theSeconds ) public Duration( int totalSeconds ) public int getHours( ) public int getMinutes( ) public int getSeconds( ) public int getTotalSeconds( ) public Duration add( Duration aDuration ) public Duration subtract( Duration aDuration ) public String toString( ) private String padLeadingZero( int number )

La classe Duration permet de stocker la durée d'une piste et d'additionner deux durées. En effet, les méthodes **add** et **subtract** sont sans doute les plus intéressantes de la classe.

La méthode **add** est capable d'additionner la durée de l'objet courant avec celle passée en paramètre. Le résultat de l'addition est renvoyé par la méthode.

🔗 Créez la classe **Duration** ainsi qu'un programme de test appelé **TestDuration** où vous pourrez vérifier le bon fonctionnement de la classe .La documentation complète de la classe est à l'adresse : <http://gita-server/java>

**c. Classe « Piste »**

<b>Piste</b> { From act5 }
<i>Attributes</i>
private String nom
<i>Operations</i>
public Piste( String nom, Duration duree )
public void setNom( String nom )
public void setDuree( Duration duree )
public String getNom( )
public Duration getDuree( )

La classe Piste ne pose pas de difficulté particulière. Elle se contente d'encapsuler les données nécessaires avec des méthodes « accesseurs de données » (get et set).

🔗 Créez la classe **Piste**. Aidez-vous de la documentation sur <http://gita-server/java>

**d. Classe « Album »**


<b>Album</b> { From act5 }
<i>Attributes</i>
private String nom
private String formation
private int dateDeSortie
<i>Operations</i>
public Album( String nom, String formation, int dateDeSortie, Piste listePistes[0..*] )
public void setNom( String nom )
public void setFormation( String formation )
public void setDateDeSortie( int dateDeSortie )
public void setListePistes( Piste listePistes[0..*] )
public String setNom( )
public String getFormation( )
public int getDateDeSortie( )
public Piste[0..*] getListePistes( )
public String toString( )
public Duration calculeDuree( )

La classe `Album` ne pose pas trop de problème de conception. La méthode la plus délicate est sans doute la méthode `calculeDuree` dont le rôle est d'additionner les durées de toutes les pistes et de renvoyer un objet « `Duration` » représentant la durée totale de l'album.


La méthode `toString()` est censée renvoyer une chaîne de caractère donnant cet affichage :

```
Album : Brand new day
Nom de l'artiste : Sting
Duree totale : 00:48:49

Piste No 1 : A thousand years (00:05:57)
Piste No 2 : Desert rose (00:04:46)
Piste No 3 : Big lie small world (00:05:05)
Piste No 4 : After the rain has fallen (00:05:04)
Piste No 5 : Perfect love...gone wrong (00:05:25)
Piste No 6 : Tomorrow we'll see (00:04:47)
Piste No 7 : Prelude to the end of the game (00:00:19)
Piste No 8 : Fill her up (00:05:37)
Piste No 9 : Ghost story (00:05:29)
Piste No 10 : Brand new day (00:06:20)
```

 Créez la classe `Album` en respectant le cahier des charges. Doc sur <http://gita-server/java>

#### e. Classe « `TestAlbums` »

 Créez cette classe dont le code source est le suivant. Téléchargeable sur <http://gita-server/java/TestAlbums.java>

D:\Peda\info\prog\Java\activites\PrjActivitesJava\src\org\diderot\act5\TestAlbums.java

```
1 package org.diderot.act5;
2
3 import javax.swing.*;
4
5
6 public class TestAlbums {
7
8
9 public static void main ( String args[] )
10
11     {
12         Piste[] maListe = {
13             new Piste ("A thousand years" , new Duration (0,5,57)) ,
14             new Piste ("Desert rose" , new Duration (0,4,46) ) ,
15             new Piste ("Big lie small world" , new Duration (0,5,5) ) ,
16             new Piste ("After the rain has fallen" , new Duration (0,5,4) ) ,
17             new Piste ("Perfect love...gone wrong" , new Duration (0,5,25) ) ,
18             new Piste ("Tomorrow we&apos;ll see" , new Duration (0,4,47) ) ,
```

```

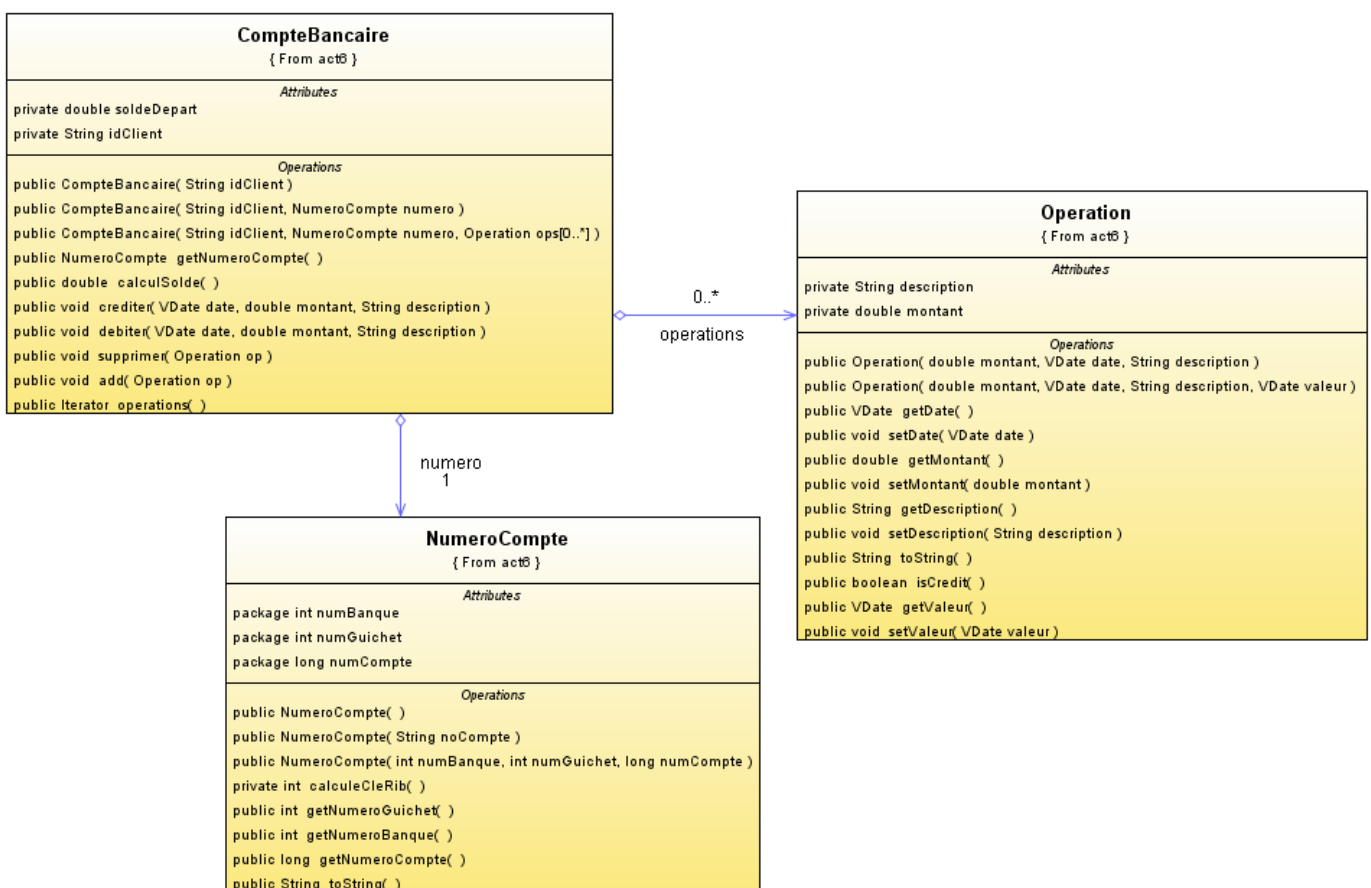
19         new Piste ("Prelude to the end of the game" , new Duration (0,0,19)) ,
20         new Piste ("Fill her up" , new Duration (0,5,37)) ,
21         new Piste ("Ghost story" , new Duration (0,5,29)) ,
22         new Piste ("Brand new day" , new Duration (0,6,20))
23
24
25         };
26
27
28     Album monAlbum = new Album ( "Brand new day" , "Sting" , 1999 , maListe);
29
30     System.out.println(monAlbum);
31
32 }
33 }
34

```

## C.6. Agrégation et Collections : Gestion de compte bancaire

### a. Objectifs

L'objectif de cette activité est la maîtrise de la relation d'agrégation entre 2 classes en utilisant une collection. Ces relations « navigables » définissent plusieurs agrégations entre les classes **CompteBancaire**, **NumeroCompte** et **Operation**.




### b. Classe « NumeroCompte »

Le code source et la doc de cette classe sont fournis sur <http://gita-server/java>

**c. Classe « Operation »**

Cette classe est chargée de gérer une opération bancaire qu'elle soit au crédit ou au débit.

 Codez la classe *Operation* en respectant le cahier des charges (Documentation de la classe)

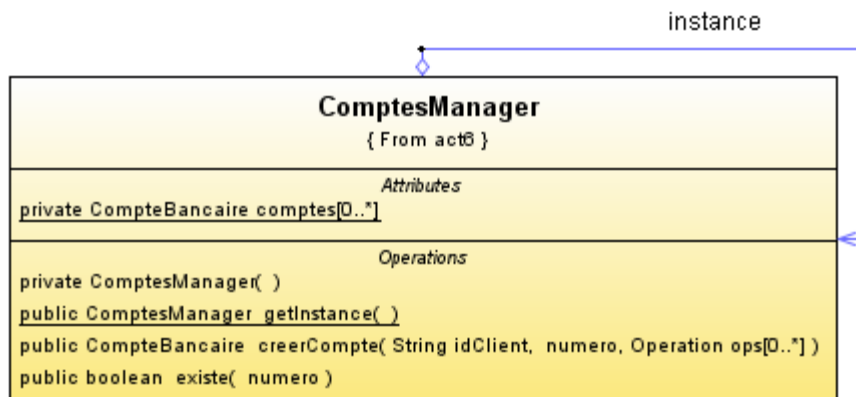
**d. Classe CompteBancaire**

Sans doute la classe la plus complexe, la classe « *CompteBancaire* » permet de:

- Ajouter et supprimer des opérations
- Créditer ou débiter
- Calculer le solde du compte

**e. Gestionnaire de comptes bancaires : Classe ComptesManager**

Afin de garantir qu'aucun compte bancaire ne puisse être dupliqué ou ne puisse avoir un numéro de compte existant déjà, on choisit de créer une classe capable de gérer la création des comptes et leur unicité. Ce modèle de conception (Design Pattern) s'appelle le « Singleton » :



Dans notre cas, le seul moyen de créer un compte bancaire est d'obtenir une instance du gestionnaire en passant par la méthode « *getInstance()* ». Cette instance sera unique à chaque appel à cette méthode.

```
ComptesManager manager = ComptesManager.getInstance ();
```


Ensuite, il est relativement facile d'appeler la méthode « *creerCompte(...)* » pour créer un nouveau compte :

```
ComptesManager manager = ComptesManager.getInstance ();
```

```
Operation [] listOps = {
    new Operation(6412.0, new VDate(1, 1, 2007), "VIREMENT"),
    new Operation(-123.4, new VDate(1, 1, 2007), "Courses Auchan"),
    new Operation(-4245.12, new VDate(2, 1, 2007), "Impots")
};


NumeroCompte numero = new NumeroCompte(30004, 376, 123456789);

CompteBancaire cpt1 = manager.creerCompte("123123645", numero, listOps);
```

 Créez la classe « *ComptesManager* » en respectant le cahier des charges. La documentation complète est dispo sur <http://gita-server/Java>

### **f. Test de l'ensemble**

Il faut maintenant créer un programme de test permettant de valider l'ensemble des classes. Ce programme devra créer un compte bancaire, ajouter des opérations au crédit et au débit et afficher le solde du compte.

 Créez la classe principale « *org.diderot.act6.Main* » et validez les classes.