COAXYS

Java

Les bases

coaxys

EXPERTISE • ENGAGEMENT • TRANSPARENCE

Êtes-vous en bonne condition ?

Quel rythme adoptons nous ?

Gardons cette formation informelle !

Profitons du partage d’expérience …

Règles d’engagement

Animateur

Anthony PERIQUET

- Parcours :

  Développeur NTIC Chez des éditeurs parisiens – 6 ans
  Développeur et responsable de pôle dans le nucléaire – 4 ans

- Passions :

  Le sport, le cinéma, ...

- Aujourd’hui :

  Scrum Master
  Développeur
  Et formateur

- Ce que ce n'est pas :
  - Une danse
  - Une île
- C'est un langage de programmation :
  - Créé en 1995 par 2 employés de Sun Microsystems
  - Signifie « café » en argot américain

- Objectif portabilité
- Orienté objet
- Garbage collector : Permet de limiter les fuites de mémoire

- Deux types de JVM disponibles :
  - JRE (Java Runtime Environment)
  - JDK (Java Development Kit)
- Trois types d'environnements :
  - J2SE : Application « client lourd » (Word, Excel, …)
  - J2EE : Application « client léger » (Site web, …)
  - J2ME : Application portable (mobile, PDA, …)
- Téléchargement :
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html

- Définition : Integrated Development Environment
  - Disponible pour un ou plusieurs langages de programmation (**Java**, .Net, PHP, Python, …)
- Quelques exemples d'IDE pour Java :
  - Eclipse
  - IDEA IntelliJ
  - NetBeans
  - *Android Studio*

- Il est composé d'au moins une classe
- Il doit contenir la méthode *main*

```
public class test {
	public static void main(String[] args) {
		// TODO : Insert content
	}
}
```

- Et c'est à peu près tout :)

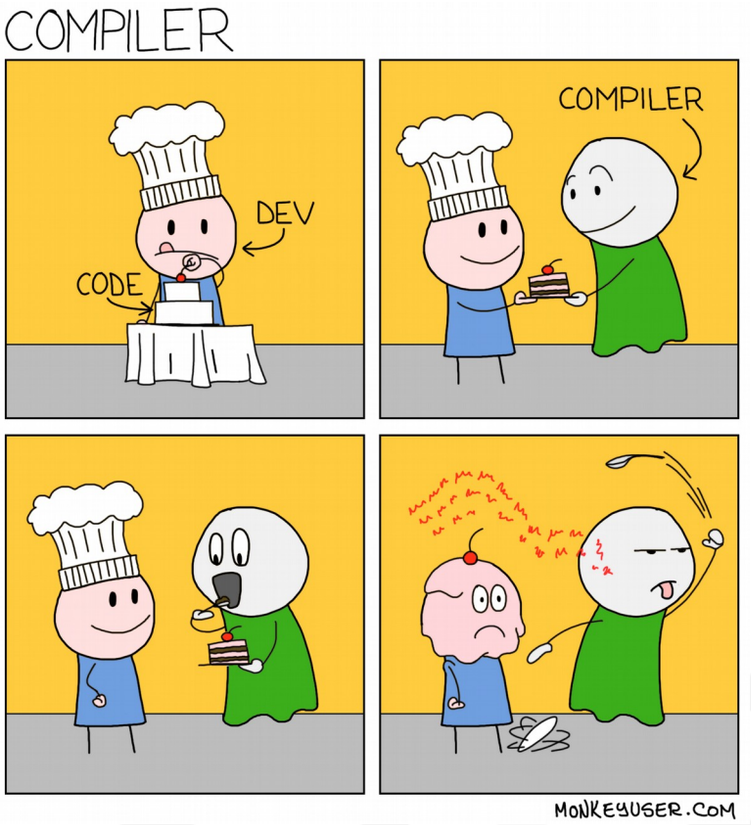
- Les sorties standard
  - System.out.print
  - System.out.println
- Les entrées standard
  - C’est un peu plus compliqué mais pas trop
  - Coming soon
- Les commentaires

```
// Ceci est un commentaire
/* Ceci est un commentaire */

/*
 * Ceci est aussi un commentaire
 */
```

Ça sert à quoi ?

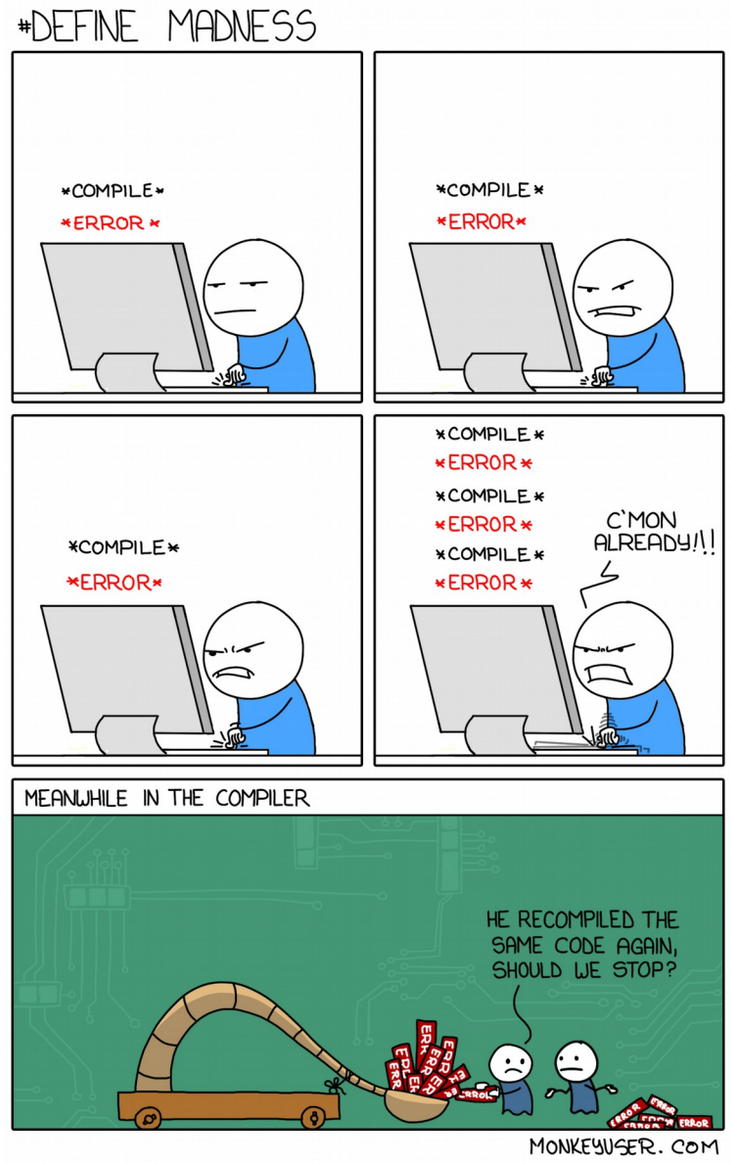
- A générer du bytecode (fichiers précompilés)
- Nettoyer le code des notions inutiles : Vous avez dit commentaires ?



- Pour compiler on utilise l'utilitaire *javac*
- ... avec un fichier dont l'extension est *.java*

```
$> javac MaClasse.java
```

```
# Bonus
$> docker run --rm \
    -u $(id -u):$(id -g) \
    -v $(pwd):/app \
    openjdk \
    javac /app/TP1.java
```



- Créer un programme qui affiche à l’écran le texte *Hello World !*
- Créer un programme qui affiche à l’écran le texte *Hello* "*World*" *!*

Nom	Taille en octets lors des calculs	Valeur par défaut
boolean	Un seul bit suffit, mais on réserve souvent un octet pour les stocker.	false
byte	1	0
short	2	0
int	4	0
long	8	0
char	2	'\u0000'
float	4	0.0
double	8	0.0
Object	Dépendant de la machine virtuelle	null

- Il existe d'autres types qui dérivent du type *object*
  - Exemple : *String*
- On peut également créer des tableaux avec les types primitifs
  - Exemple : On déclare un tableau de 10 entiers

```
int[] tab = new int[10] ;
```

```
int tab[] = new int[10] ;
```

```
int tab[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} ;
```

Créer un programme qui :

- Utilise une variable dans laquelle on stocke un prénom
- Affiche le message *Bonjour [prénom]*

Créer un programme qui :

- *Divise deux entier et affiche le résultat de l'opération*

- En théorie

```
Scanner scan = new Scanner(System.in);
String input = scan.nextLine();
```

- En pratique :
  - Créer un programme qui demande le nom de l’utilisateur et lui dit bonjour.

- La structure if ... else

```
if ( condition ) {
      // TODO : Faire quelque chose si la condition est réalisée
} else {
      // TODO : Faire autre chose sinon
}
```

- La structure switch

```
switch( variable ) {
      case value1 : return 1 ;
      case value2 : return 2 ;
      default : return null ;
}
```

- Pour qu'une condition soient évaluée, elle nécessite un ou plusieurs opérateurs logiques

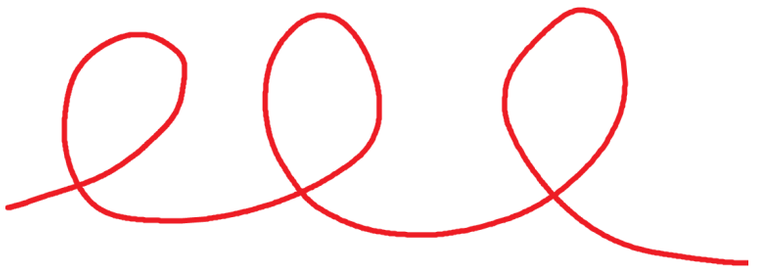
Opérateur	Effet
==	Teste l'égalité
!=	Teste la différence
<	Teste l'infériorité stricte
<=	Teste l'infériorité ou l'égalité
>	Teste la supériorité
>=	Teste la supériorité ou l'égalité
&&	ET - Permet de préciser une condition
||	OU – Permet de préciser une condition
?	Opérateur ternaire

Écrire un programme qui demande à l'utilisateur de saisir son âge et afficher à l'écran à quelle catégorie de personne il correspond selon les catégories suivantes :

- Bébé (0 – 2)
- Enfant (3 – 12)
- Adolescent (13 – 18)
- Adulte (19 – 60)
- Sénior (60+)

- **while** : Tant que ma condition est valide, je répète

```
while ( condition ) {
	// TODO : Faire quelque chose
}
```

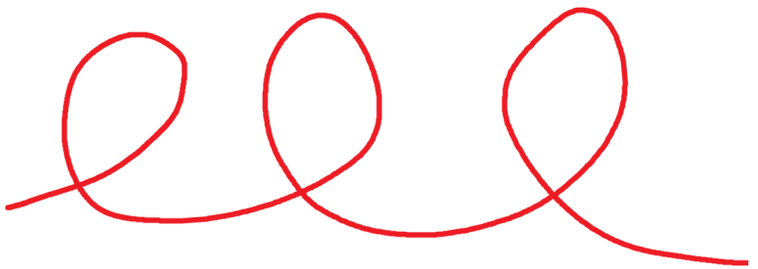


- Écrire un programme qui demande à l'utilisateur de saisir son âge et afficher à l'écran à quelle catégorie de personne il correspond selon les catégories suivantes :
  - Bébé (0 – 2)
  - Enfant (3 – 12)
  - Adolescent (13 – 18)
  - Adulte (19 – 60)
  - Sénior (60+)
- Le programme demande à l'utilisateur s'il souhaite recommencer sa saisie. Si l'utilisateur accepte, le programme recommence sinon il se termine.

- **do … while** : Je fais quelque chose et, si ma condition est valide, je recommence.

```
do {
	// TODO : Faire quelque chose
} while ( condition ) ;
```

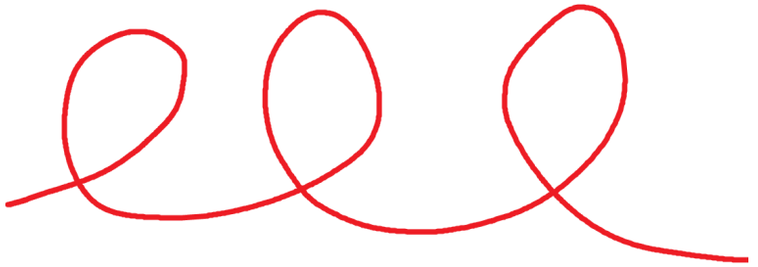
Cette boucle s’exécute au moins une fois.



Écrire un programme qui demande à l'utilisateur de saisir un nombre.
La saisie s'arrête quand l'utilisateur renseigne la valeur 0.

Lorsque l'utilisateur a saisi 0 :

- Afficher la plus petite valeur saisie.
- Afficher la plus grande valeur saisie
- Afficher la moyenne de toutes les valeurs saisies



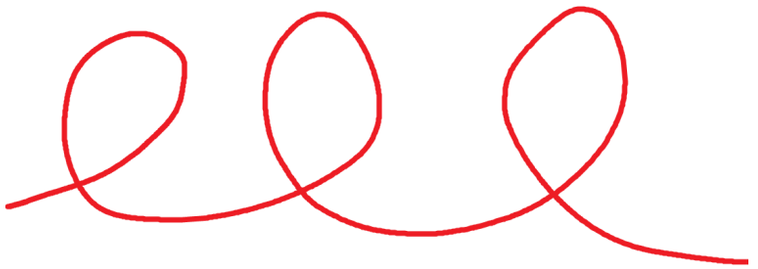
- **for** : C'est un boucle particulière qui itère sur quelque chose qui est défini dans sa propre condition.

```
for (int i = 0 ; i <= 10 ; i++) {
	// TODO : Faire quelque chose
}
```

- On peut le lire comme suit :

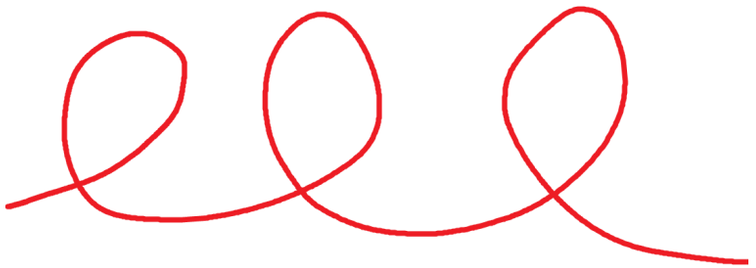
```
pour (prédicat de départ; condition d'itération; valeur de l'itération) {
	// TODO : Faire quelque chose
}
```

- En français : Pour i (entier) allant de 0 à 9 (inférieur ou égal à 10) je fais quelque chose et j'ajoute 1 à i à la fin de chaque itération

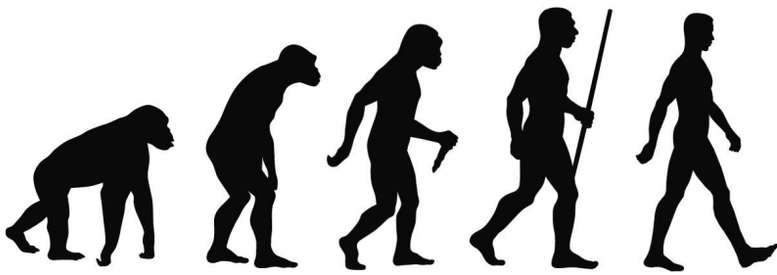


Demander à l’utilisateur de saisir un nombre et définir s’il s’agit d’une nombre premier. Lui demander ensuite s’il veut effectuer une nouvelle saisie.

Rappel : Un nombre premier n’est divisible que par 1 et par lui-même.



- Une méthode est une fonction de classe
- Rôle bien spécifique
- Structure le code, le rend atomique
- Quelques exemples :
  - String.toLowerCase()
  - String.length()
  - System.out.println()
  - main()

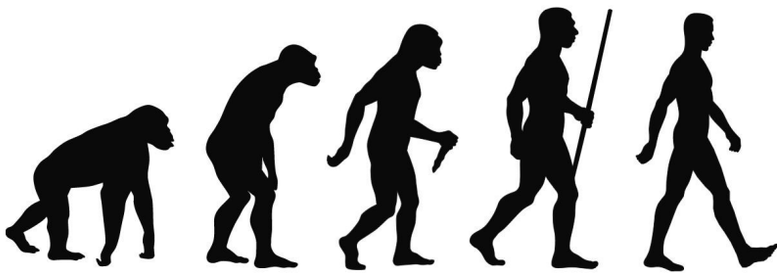


- Syntaxe :

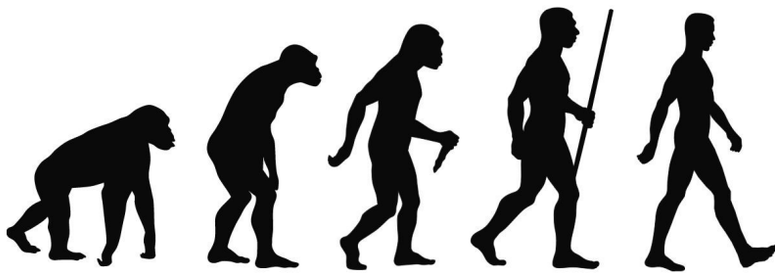
```
type_retour nom_fonction ([type_paramètre nom_paramètre, ...]) {
	// TODO : Contenu de la fonction
}
```

- Exemple

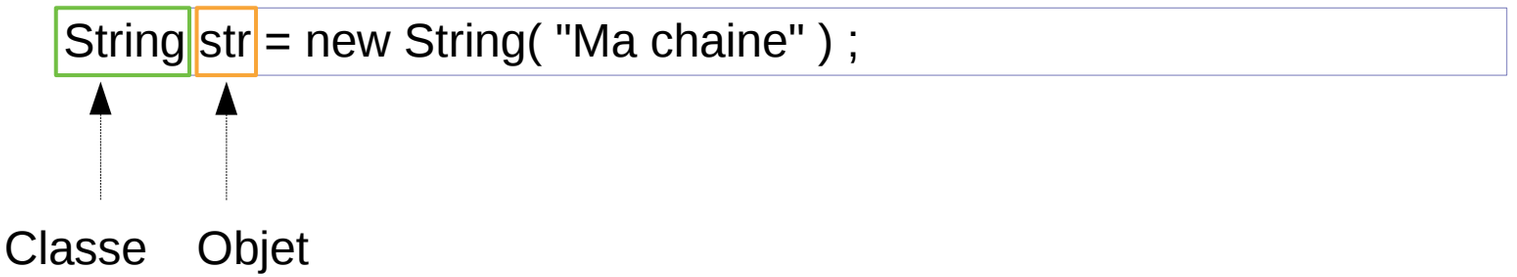
```
int square(int nb) {
	return nb * nb ;
}
```



Reprendre le TP sur les nombres premier et isoler les parties qui peuvent l’être



- Deux notions importantes :
  - Les classes : Modèle ou masque
  - Les objets : Instance



- Masque général d'un objet
- Définit le comportement ...
- ... et les propriétés

```
class MaClasse {
	int property1 ;
	int property2 ;

	public void method1() {
		// Comportement attendu
	}
}
```

- Le constructeur

```
class MaClasse {
	int property1 ;
	int property2 ;

	public MaClasse() {
		// Comportement attendu
	}
}
```

Pas de type de retour

Créer un programme qui créé un objet stagiaire définit comme suit :

- Nom
- Prénom
- Age

L’utilisateur saisit les informations qui définissent l’objet
Le programme affiche ensuite la fiche d’identité du stagiaire

- Les variables de classes sont définies avec le mot clé static

```
class MaClasse {
	static int property2 ; // Variable de classe
}
```

- Les constantes sont définies avec le mot clé final

```
class MaClasse {
	static final int property3 = 0 ; // Constante de classe, valeur figée
}
```

- Quelques notions d'architecture de projet
  - Le projet
  - Le package
  - La classe

```
MonProjet /
    └ Package1
    ┆   └ MaClasse1.java
    └ Package2
    ┆   └ Package3
    ┆   ┆   └ MaClasse2.java
    ┆   └ MaClasse3.java
```

- Les mots clés :
  - package permet de définit le package dans lequel se trouve la classe
  - import permet de référencer une classe d'un autre package afin de pouvoir l'utiliser

```
package package1;

import package2.MaClasse3;
import package2.package3.MaClasse2 as AliasClasse;

class MaClasse1 {
    static int property2; // Variable de classe
}
```

- Visibilité ou portée
  - public
  - protected
  - private
- S’applique aux classes, méthode et attributs
- this

```
class MaClasse {
    private int property1;

    public MaClasse(int value) {
        this.property1 = value;
    }
}
```

- Créer une classe Personne définie par :
  - Un nom (privé)
  - Un prénom (privé)
  - Un email (privé)
- La classe Personne est capable de dire combien d'objets ont été créés
- Le constructeur de la classe prend les paramètres suivants :
  - Nom
  - Prénom
- Le mail est défini comme suit : nom.prenom@example.com
- Demander à l'utilisateur d'ajouter des personnes

- Permet de défini plusieurs méthodes avec le même nom ...
- ... mais une signature différente

```
public String bonjour() {
    return "Bonjour";
}
```

```
public String bonjour(String name) {
    return "Bonjour " + name;
}
```

Le nombre et / ou le type des arguments doit être différent.

- Une notion fondamentale de la programmation objet
- Évite la redondance
- Permet d'améliorer une classe
  - En apportant de nouvelles propriétés
  - En apportant de nouvelles méthodes
  - En apportant des règles métier
- extends permet d'étendre une classe
- super permet d'accéder à la classe mère

⚠ L'héritage multiple est impossible

```
class Parent {
    String property1;

    public Parent(String value) {
        this.property1 = value;
    }
}
```

```
class Enfant extends Parent{
    String property2;

    public Enfant(String value, String value2) {
        this.property1 = value;
        This.property2 = value2;
    }
}
```

- D’une classe mère à une classe fille
- Signature identique …
- … mais traitement différent

Ne pas confondre avec la surcharge

- Créer un programme qui permet de gérer des œuvres papier (50 max) :
  - Chaque œuvre est identifiée par un titre, un auteur et un genre
  - Le programme permet de saisir des œuvres qui sont des romans, des Bds, des manuels ou des livres de recette (autres si vous le souhaitez)
  - Le programme peut afficher les œuvres enregistrées (fiche détaillée)
  - Le programme permet de connaître la répartition des œuvres selon le genre
  - Un roman peut avoir un format (broché, poche, e-book)
  - Une BD peut être en couleur ou en noir et blanc
  - Un livre de cuisine peut être illustré

- abstract permet de déclarer une classe comme Abstraite
- Une classe abstraite ne peut être instanciée
- Héritage, polymorphisme et abstraction sont des notions fortement liées
- Force l'implémentation des méthodes

```
abstract class Humain {
    abstract public void travailler();
}

class Formateur extends Humain{
    public void travailler() {
        System.out.println("Enseigne") ;
    }
}

class Stagiaire extends Humain{
    public void travailler() {
        System.out.println("Apprend") ;
    }
}
```

- Ce sont des classes 100 % abstraites
- Elles représentent un contrat d'implémentation
- Elle n'ajoute aucune propriété à moins qu'elle ne soit statique
- interface permet de déclarer une interface
- implements permet d'implémenter une interface

```
public interface Developer {
    public void program();
}

class Worker implements Developper{
    public void program() {
        System.out.println("Créé des programmes");
    }
}
```

Créer un programme qui permet de gérer le salaire des employés d'une entreprise de 30 personnes.

Un employé est défini par son nom, prénom, âge et date d'embauche.
La classe Employe dispose d'une méthode qui retourne « [Prénom] [NOM] » .

Le calcul du salaire dépend du type de l'employé :

- Vente : 20 % du chiffre d'affaire mensuel + 400€
- Représentation : 20 % du chiffre d'affaire mensuel + 800€
- Production : Nombre d'unités produites multiplié par 5
- Manutention : Nombre d'heure effectuée multiplié par 15

Selon le type d'employé, la méthode qui retourne le nom est suffixée par la catégorie de l'employé.

Certains employé de production ou de manutention bénéficient d'une prime de risque mensuelle de 200€. Il appartiennent à une catégorie à part d'employés.

Après avoir renseigné les effectifs de l'entreprise, le programme restitue l'identité et le salaire de chacun des employés.

- Ce sont des tableaux à taille variable
- Il en existe plusieurs types :
  - Les listes
  - Les sets
  - Les maps

- Les listes sont des tableaux à taille variable
- Il en existe plusieurs implémentations :
  - LinkedList
  - ArrayList

```
ArrayList l = new ArrayList() ;
l.add("Bonjour") ;
l.add(12) ;
l.add(new Object()) ;
```

- Les listes implémentent l'interface Iterable

- Contrainte d'unicité
  - Impossible d'avoir deux fois le même élément
- Plusieurs implémentations :
  - HashSet
  - TreeSet
  - LinkedHashSet

```
HashSet s = new HashSet() ;
s.add("Bonjour") ;
s.add(12) ;
s.add(new Object()) ;
```

- Collection de type clé – valeur
  - La clé sert d'index
- Consommateur en mémoire

```
Map<String, String> m = new HashMap() ;
m.put("key1", "Bonjour") ;
m.put("key 2", "Le monde") ;
```

Reprendre le TP précédent et le modifier comme suit :

Mettre en place une classe Personnel qui permet d'ajouter des employés, quel que soit leur type. Cette classe doit disposer des méthodes suivantes :

- void ajouterEmployer(Employe)
- doid calculerSalaires()
- double salaireMoyen()

- Erreur qui conduit le plus souvent à l'arrêt du programme
- Une exception est une instance de classe
  - Il en existe plusieurs implémentations (IOException, …)
  - Elles peuvent être personnalisées (MonException)
  - La classe Exception est la classe de plus haut niveau
- Le bloc try{ … } catch{ … } permet de gérer les exceptions

```
try {
	System.out.println("Resultat : " + (1/0)) ;
} catch (ArithmeticException e) {
	e.printStackTrace() ;
} finally {
	System.out.println("Action systématique") ;
}
```

Reprendre le TP précédent et y intégrer la gestion d’exceptions nécessaire afin de valider :

- Les bonnes saisies
- Les erreurs d’implémentation

- On y accède avec l'objet File

```
File file = new File("/path/to/file.ext") ;
```

- Il permet de manipuler le fichier

```
System.out.println( "Le fichier existe : " + file.getName() ) ;
System.out.println( "Nom de fichier : " + file.getName() ) ;
if (file.isDirectory()) {
	File[] files = file.listFiles() ;
} else {
	file.delete() ;
}
```

Créer un programme qui liste affiche les informations relative à un chemin spécifié par l'utilisateur

- Si le chemin spécifié est un fichier, il faudra afficher les permissions, la taille (de manière intelligible) et le nom du fichier.

- Si le chemin spécifié est un répertoire, il faudra en afficher la liste des fichiers (cf. instruction précédente) et afficher la taille totale de ces fichiers.

- Pour agir sur le contenu d'un fichier nous utilisons 2 types de flux :
  - InputStream pour la lecture (ex : FileInputStream)

```
FileInputStream input = null;

try {
    input = new FileInputStream( new File( "/path/to/file" ) );
    byte[] buffer = new byte[8];       // On défini la taille du buffer

    while( input.read(buffer) ) {      // Tant qu'on peut lire le fichier ...
        for (byte bit : buffer) {      // On affiche son contenu
            System.out.print( (char)bit );
        }
        buffer = new byte[8] ;         // On réinitialise afin d'éviter les résidus
    }
} catch (FileNotFoundException | IOException e) {
    e.printStackTrace();
} finally {
    if (input != null){ input.close; }   // Devrait faire l'objet d'un try-catch
}
```

- Pour agir sur le contenu d'un fichier nous utilisons 2 types de flux :
  - OutputStream pour l'écriture (ex : FileOutputStream)

```
FileInputStream input = null;
FileOutputStream output = null;

try {
    input = new FileInputStream( new File( "/path/to/file" ) );
    output = new FileOutputStream( new File( "/path/to/file_copy" ) );
    byte[] buffer = new byte[8];       // On défini la taille du buffer

    while( input.read(buffer) ) {      // Tant qu'on peut lire le fichier ...
        output.write( buffer );        // On écrit dans le second fichier
        buffer = new byte[8];          // On réinitialise afin d'éviter les résidus
    }
} catch (FileNotFoundException | IOException e) {
    e.printStackTrace();
} finally {
    if (input != null){ input.close; }   // Devrait faire l'objet d'un try-catch
    if (output != null){ output.close; }
}
```

- Il est possible d'utiliser des flux avec un tampon de lecture
  - Cela augmente grandement les performances

```
BufferedInputStream input = null;
BufferedOutputStream output = null;

try {
    input = new BufferedInputStream(new FileInputStream( new File( "/path/to/file" ) );
    output = new BufferedOutputStream(new FileOutputStream( new File( "/path/to/file_copy" ) );
    byte[] buffer = new byte[8];

    while( input.read(buffer) != -1 ) {
        output.write( buffer );
        buffer = new byte[8];
    }
} catch (FileNotFoundException | IOException e) {
    e.printStackTrace();
} finally {
    if (input != null){ input.close; }
    if (output != null){ output.close; }
}
```

- On peut écrire des objets dans un fichier.

- L'objet doit implémenter l'interface Serializable

```
public class PersistantData implements Serializable {
    private String label;
    private int number;

    public PersistantData( String Label, int number) {
        this.label = label;
        this.number = number;
    }

    public String toString() {
        return "My data n° " + this.number + " : " + this.label;
    }
}
```

- transient permet d'ignorer une propriété pour la sérialisation

Écrire un programme qui permet de gérer une médiathèque dont les objets suivants en seront les pierres angulaire :

- Media
- Catégorie
- Acteur
- Réalisateur
- Genre

Le programme disposera d’un mode édition et consultation et ses données seront sauvegardées afin de pouvoir être rechargées à la prochaine exécution.

;#JOIN US;

coaxys

Coaxys est toujours en mouvement, découvrez toute notre actualité ...

2 ter rue Georges Charpak, 76130 Mont Saint Aignan
contact@coaxys.com
www.coaxys.com

Merci !