COURS DE JAVA ORIENTÉ OBJET

Présenté par : Amadou SEYDOU
Ingénieur logiciel

Objectifs du cours

À l'issue de ce cours, l'étudiant devrait

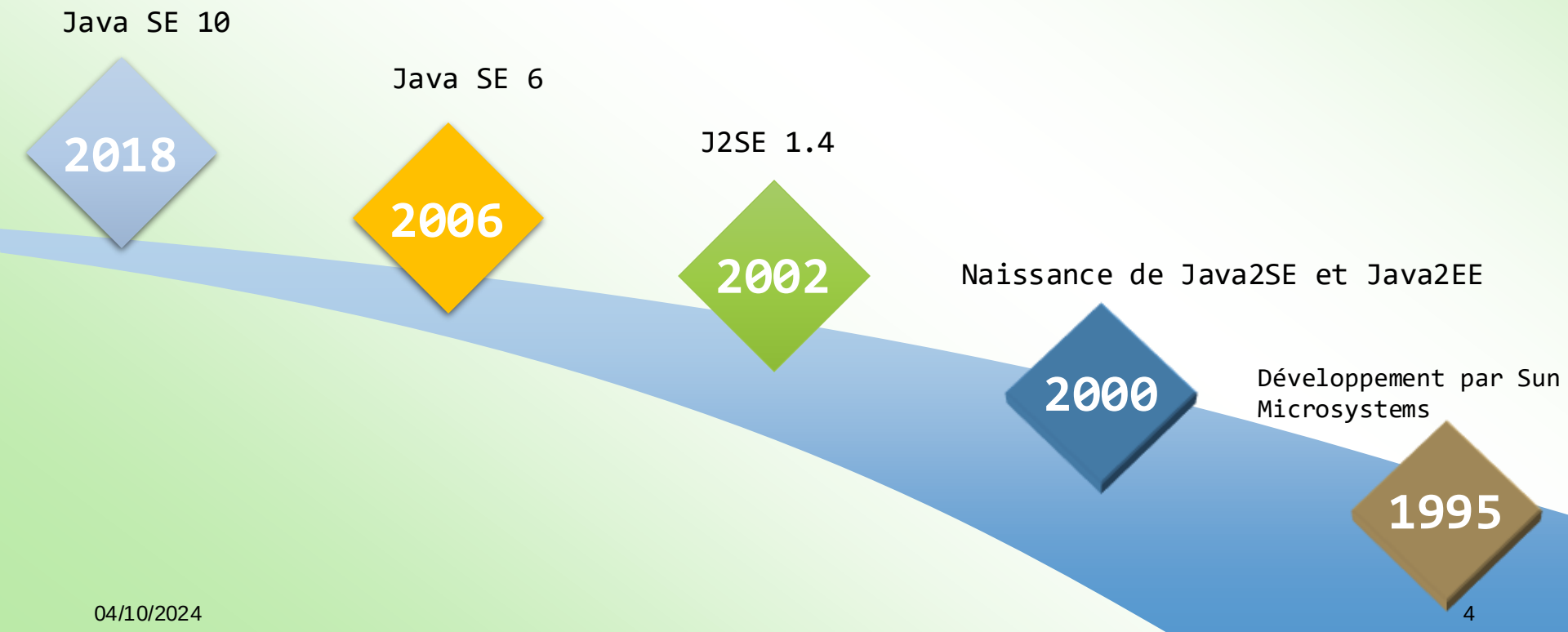
- Maîtriser les notions de base de Java
- Maîtriser la notion de *classe* et des *objets* en Java
- Comprendre le principe d'héritage et des interfaces
- Maîtriser le principe des exceptions en Java
- Manipuler les fichiers

Plan du cours

- Les outils de développement
- Notions de base de Java
- Java Orienté Objet
- TP

Un peu d'historique …

- Réutilisable : orienté objet simple
- Portabilité : indépendant des machines
- Robustesse : performant



Fonctionnement de Java

Les bibliothèques logicielles

- **Java Virtual Machine (JVM)**

  La machine virtuelle sur laquelle s'exécute un bytecode Java

- **Java Runtime Env (JRE)**

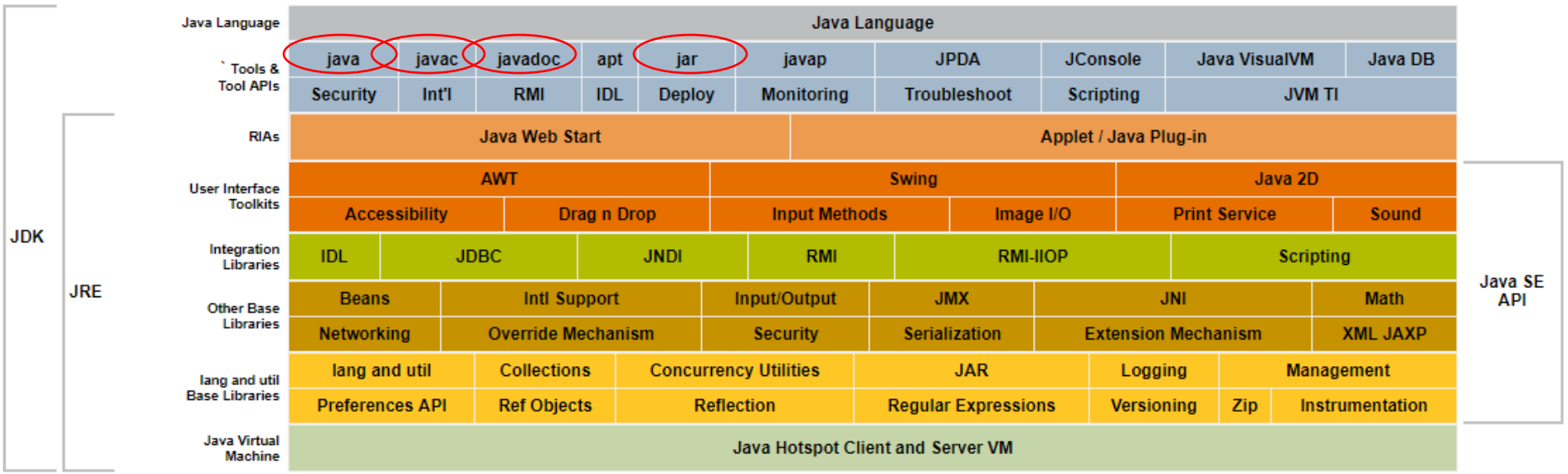
  Ensemble des logiciels qui permettent l'exécution d'un programme Java

- **Java Development Kit (JDK)**

  Ensemble de technologies comprennant un JRE et autres programmes permettant l'exécution d'un programme JAVA

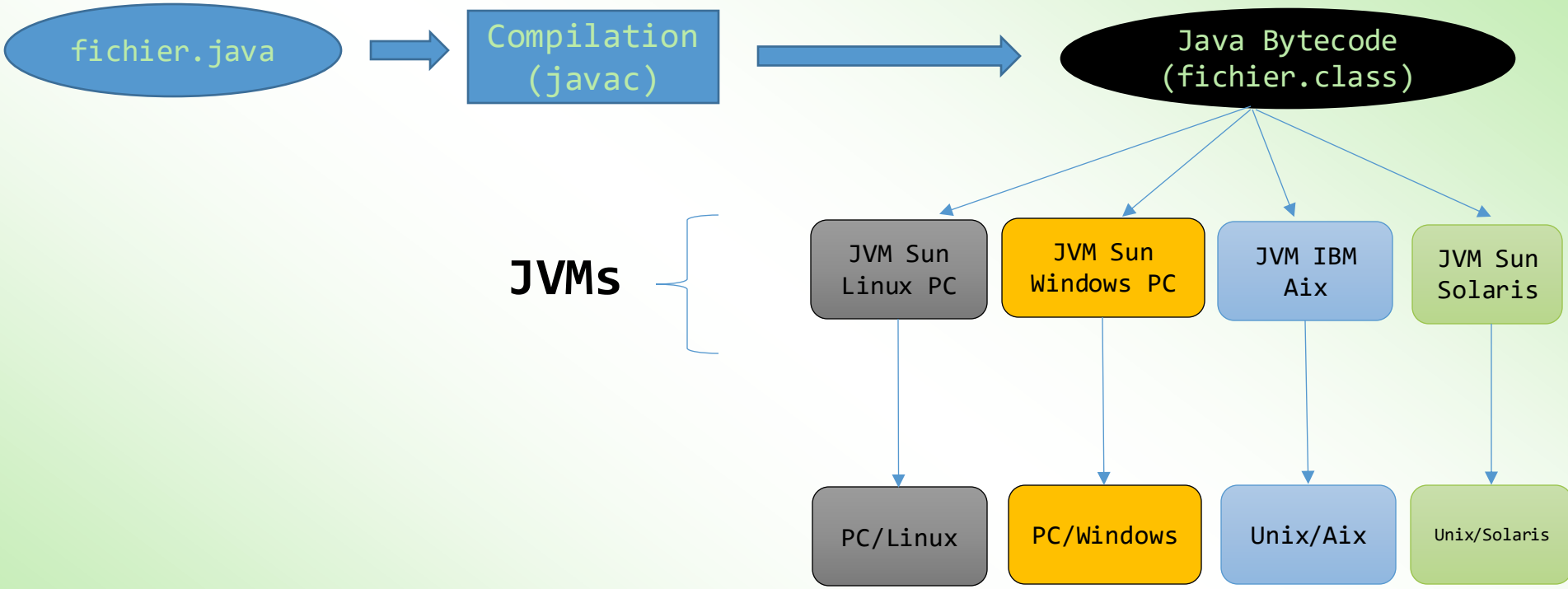
Fonctionnement de Java

Les bibliothèques logicielles



Fonctionnement de Java

Processus d'exécution d'un programme Java



OUTILS DE DEVELOPPEMENT

Outils de développement

Installation des environnements

Outils de développement

Eclipse

- Logiciel libre, extensible et polyvalent : VS Code
- Environnement de développement intégré
- Gère plusieurs langages de programmation dont Java évidemment!
- Téléchargeable sur https://code.visualstudio.com/

NOTIONS DE BASE DE JAVA

NOTIONS DE BASE DE JAVA

Liens utiles pour développer en Java

- https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java?status=published
- https://java.developpez.com/livres-collaboratifs/javaenfants/

NOTIONS DE BASE DE JAVA

Déclaration des variables

- **En Java, on a deux types de variables :**
  - ✓ **Variable simple (ou primitive)**
  - ✓ **Variable complexe (ou objet)**

```
<type de la variable> <nom de la variable>;
```

```
int  ma_variable = 1;
```

OU

```
int  ma_variable;
ma_variable = 1;
```

NOTIONS DE BASE DE JAVA

La fonction *main*

- **Permet d'exécuter des programmes Java**

```
public static void main(String[] args) {
        // Affiche un message "Hello les M1 J2M.".

        System.out.println("Hello les M1 J2M.");
}
```

NOTIONS DE BASE DE JAVA

Les packages

- **Ensemble de librairies utiles dont dispose l'utilisateur pour manipuler les fonctionnalités Java**

  - ✓ Pour lire/écrire des fichiers, on utilise le package **java.io**
  - ✓ Pour faire des calculs mathématiques, on utilise le package **java.Math**
  - ✓ Pour recupérer une variable saisie par l'utilisateur, on utilise **java.util.Scanner**
  - ✓ etc

- **Importer un package en Java dans votre programme**

```
import java.io;
```

NOTIONS DE BASE DE JAVA

Les entrées/sorties

- **Entrées**

C'est que l'utilisateur saisit depuis son ordinateur : **clavier**

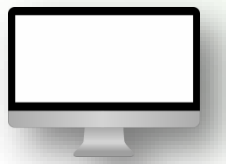
En Java, on utilise une classe **Scanner du package java.util.Scanner** pour lire les données venant de son clavier

➢ Pour lire les données du clavier :

```
Scanner sc = new Scanner(System.in);
String saisie = sc.nextLine();
```

- **Sorties**

C'est que l'application affiche à l'utilisateur : **écran**



➢ Pour afficher un message à l'utilisateur :

```
System.out.println("Affiche un message à l'utilisateur");
```

NOTIONS DE BASE DE JAVA

Exercice d'application N°0a

Écrire un programme qui demande à un utilisateur de saisir une variable et d'afficher la variable saisie par l'utilisateur.

Exemple :

Entrées :

Veuillez saisir une valeur :

(L'utilisateur saisit une valeur au clavier, par exemple 12)

Sorties :

Vous avez saisi la valeur 12.

NOTIONS DE BASE DE JAVA

Exercice d'application N°0b

Écrire un programme qui demande à un utilisateur de saisir deux variables et d'afficher la **somme de celles-ci.**

Exemple :

Entrées :

Veuillez saisir la première valeur :

(L'utilisateur saisit une valeur au clavier, par exemple 12)

Veuillez saisir la seconde valeur :

(L'utilisateur saisit une valeur au clavier, par exemple 5)

Sorties :

Vous avez saisi les valeurs 12 et 5 et leur somme est 17.

NOTIONS DE BASE DE JAVA

Les conditions

```
if(//ma_condition)
{
	// Exécuter les instructions si la condition est VRAIE
}
else if(//ma_condition)
{
	// Exécuter les instructions si cette condition est VRAIE et
la première FAUSSE
}
else
{
	// Exécuter les instructions si aucune des deux conditions
n'est VRAIES
}
```

NOTIONS DE BASE DE JAVA

Les opérateurs de comparaison

- == : Compare deux valeurs et vérifie si **elles sont égales**
- <= : Vérifie si une valeur est **inférieure ou égale** à une autre
- < : Vérifie si une valeur est **strictement inférieure** à une autre
- >= et > : Pareil que précedément mais **supérieure** à une autre valeur
- != : Vérifie si une valeur est **différente d'une autre**
- % : Calcule le **reste de la division de deux valeurs**

```
int a = 2;
int b = 3;

if( a <= b )
{
  System.out.println("La variable " + a + " est inférieure à " + b);
}
else
{
  System.out.println("La variable " + a + " est supérieure à " + b);
}
```

NOTIONS DE BASE DE JAVA

Exercice d'application N°0c

Écrire un programme qui demande à un utilisateur de saisir deux variables et d'afficher le **plus grand de celles-ci.**

Exemple :

Entrées :

Veuillez saisir la première valeur :

(L'utilisateur saisit une valeur au clavier, par exemple 12)

Veuillez saisir la seconde valeur :

(L'utilisateur saisit une valeur au clavier, par exemple 5)

Sorties :

Vous avez saisi les valeurs 12 et 5 et le plus grand est 12.

NOTIONS DE BASE DE JAVA

Les opérateurs logiques

- **ET : && (retourne *true* si les 2 conditions valent *true*, ou *false* sinon)**
- **OU : || (retourne *true* si au moins une des 2 conditions vaut *true*)**
- **NON : ! (retourne *true* si la variable vaut *false*, et *false* si elle vaut *true*)**

```
int a = 2;
int b = 3;
int c = 7;
int d = a*b;

if( !(a <= b && c%2 == 0 || d == 6))  FAUX
{
  System.out.println("La variable" + c + " est multiple de 2");
}
else if(a > b)  FAUX
{
  System.out.println("La variable" + b + " est bien inférieure à " + a);
}
else
{
  System.out.println("La variable" + b + " est bien supérieure à " + a);  ✓ OK
}
```

NOTIONS DE BASE DE JAVA

Exercice d'application N°1

Faire un programme qui permet de savoir si **une année est bissextile.**

- L'application demande à un utilisateur de saisir une année
- L'utilisateur devra saisir une année **entre 1970 et 2018.**
- Si l'utilisateur saisit une valeur en **dehors de l'intervalle,** il faut afficher un message d'erreur.
- Si l'année saisie est bonne, l'application affiche : Votre année aaaa est bissextile (ou pas).

Pour rappel :

Une **année bissextile** est une année qui **est multiple de 400.**

Et si elle **est multiple de 4,** elle ne doit **pas être multiple de 100.**

NOTIONS DE BASE DE JAVA

Les conditions

```
switch(ma_variable)
{
    case 1:
    // Exécuter la condition si ma_variable = 1
    break; // Sort du switch.
    case 2:
    // Exécuter la condition si ma_variable = 2
    break;
    default:
    // Exécuter si ma_variable n'est égale à aucune des "case"
    (exécution par défaut)
    break;
}
```

NOTIONS DE BASE DE JAVA

Exercice d'application N°2

En utilisant l'instruction ***switch*** :

L'utilisateur saisit une note des élèves sous forme de lettre :

- Si la note vaut "A", alors l'application affiche : "Excellent", votre note est "A"
- Si elle vaut "B" ou "C", alors l'application affiche : "Très bien", votre note est "B" ou "C"
- Si elle vaut "D", alors l'application affiche : "Bien", votre note est "D"
- Si elle vaut "E", alors l'application affiche : "Assez Bien", votre note est "E"
- Dans tous les autres cas, l'application devra afficher : "Passable", votre note est inférieure à "E"

NOTIONS DE BASE DE JAVA

Les boucles

```
do
{
    // Exécute les instructions si ma_condition est vérifiée
    // Exécute l'instruction au moins une fois
}
while(ma_condition)
```

ou

```
while(ma_condition)
{
        // Exécuté TANT QUE ma_condition est VRAIE
}
```

NOTIONS DE BASE DE JAVA

Exercice d'application N°3

En utilisant l'instruction ***while*** :

Reprenez l'exercice N°1 sur l'année bissextile :

- Tant que l'utilisateur n'a pas saisi une année entre 1970 et 2018 :

  Afficher : Votre année doit être comprise entre 1970 et 2018

- Si l'utilisateur a saisi une bonne année, il faut afficher le message : Vous avez saisi la bonne année et votre année est bissextile (ou pas).

NOTIONS DE BASE DE JAVA

Exercice d'application N°4

En utilisant l'instruction ***do while*** :

- Effectuer une suite de tirages de nombres entiers aléatoires jusqu'à obtenir **une suite composée d'un nombre pair suivi de deux nombres impairs** (nombre en 0 et 100)
- Afficher **les suites obtenues** avant d'avoir la bonne suite
- Afficher le **nombre de coups** qu'il a fallu pour trouver la bonne suite.

Notes : Utiliser la méthode `Math.random()` de Java pour tirer aléatoirement un nombre décimal entre 0 et 1

NOTIONS DE BASE DE JAVA

Les boucles

- On **initialise** une variable
- On effectue une **condition** sur la variable initialisée par rapport à une valeur connue
- On **incrémente** (ou **décrémente)** la valeur initialisée

```
for(int i = 2; i < 10; i++)
{
	// On initialise i à 2.
	// Exécuter les instructions TANT QUE i < 10 avec un pas de i+1 (on incrémente i de 1)
	La bouble va passer de i = 2, i = 3, i = 4, ….
}

for(int i = 10; i >= 0; i-2)
{
	// On initialise i à 10.
	// Exécuter les instructions TANT QUE i >= 0 avec un pas de i-2 (on decrémente i de 2)
	La bouble va passer de i = 8, i = 6, i = 4, ….
}
```

NOTIONS DE BASE DE JAVA

Exercice d'application N°5

En utilisant l'instruction ***for*** :

Définir une constante **PI**.

- Calculer la surface d'un cercle de rayon r (r entre 0 et 15)
- Il ne faut afficher que les surfaces des cercles de rayons **r multiples de 5** :

  Afficher : Votre rayon r est multiple de 5, sa surface est de s

Rappel :

La surface **s** d'un cercle de rayon r est : $s = \pi * r^2$

NOTIONS DE BASE DE JAVA

Les tableaux

- Tableau : liste de données regroupées dans un ensemble

```
<type_du_tableau> mon_tableau [] = { les valeurs du tableau }
```

```
int mon_tableau [] = {0, 1, 3, 6, 10, 15};
```

ou

```
int mon_tableau [] = new int[6];
mon_tableau = {0, 1, 3, 6, 10, 15};
```

ou

```
int mon_tableau [];
mon_tableau [0] = 0;
mon_tableau [1] = 1;
mon_tableau [2] = 3;
mon_tableau [3] = 6;
mon_tableau [4] = 10;
mon_tableau [5] = 15;
```

NOTIONS DE BASE DE JAVA

Les tableaux

```
String mon_tableau [] = { "chaine1", "chaine2", "chaine3", "chaine4"};

char mon_tableau [] = {'a', 'b', 'c', 'd'};
```

- Pour récupérer un élément du tableau, **l'indice commençant par 0** :

```
int un_element = mon_tableau [2]; // Affiche 'c'
```

NOTIONS DE BASE DE JAVA

Exercice d'application N°6

Ecrivez un programme qui saisit un **nombre N au clavier** et qui construit un **tableau** dont le **plus grand indice est N** et dans lequel il y a, pour chaque case d'indice i, la **somme des entiers compris entre 0 et i.**

NOTIONS DE BASE DE JAVA

Les fonctions

- Exécute une action d'un programme
- Lorsque la fonction a un ***return***, **l'exécution de cette dernière se termine immédiatement**

```
<type_de_retour> maFonction(les paramètres de la fonction séparés par des ,)
{
        // Instructions de la fonction
}
```

- Création d'une fonction qui fait la somme de deux entiers

```
int somme(int a, int b)
 {
        return a + b;
 }
```

- Appel de la fonction :

```
int s = somme(2, 3);
```

NOTIONS DE BASE DE JAVA

Exercice d'application N°7a

Reprenez **l'exercice N°1 sur l'année bissextile** :

Ecrivez une **fonction** qui **prend en paramètre une année** et détermine si elle est bissextile ou pas.

Exemple : annee = 2017

f(annee) = false

NOTIONS DE BASE DE JAVA

Exercice d'application N°7b

Ecrivez une **fonction en Java** qui prend en paramètre un **entier entre 1 et 7** et retourne une chaine de caractère du **jour de la semaine correspondant.**

Si l'utilisateur saisit une **valeur en dehors de l'intervalle**, afficher un message d'erreur.

Exemple *: f(2) = "Mardi"*

NOTIONS DE BASE DE JAVA

Exercice d'application N°7c

Ecrivez une **fonction en Java** qui prend en paramètre un **tableau d'entiers désordonné** et le **trie par ordre croissant** selon **le tri à bulles.**

Exemple *: mon_tableau = [5, 1, 0, 10, 8, 20, 52]*

tri(mon_tableau) = [0, 1, 5, 8, 10, 20, 52]

Principe du tri à bulles :

Le tri à bulles permet de parcourir un tableau, comparer deux élémens consécutifs et les permuter lorsque l'élément suivant est inférieur à son prédécesseur. ***Exemple*** : mon_tableau = [5, 1, 0, 10, 8, 20, 52]

1. étape N°1 : on compare 5 et 1, 5 > 1, on permute → **mon_tableau** = [**1**, **5**, 0, 10, 8, 20, 52]
2. étape N°2 : on compare 5 et 0, 5 > 0, on permute → **mon_tableau** = [1, **0**, **5**, 10, 8, 20, 52]
3. étape N°3 : on compare 5 et 10, 5 < 10, on laisse → **mon_tableau** = [1, 0, **5**, **10**, 8, 20, 52]
4. étape N°4 : on compare 10 et 8, 10 > 8, on permute → **mon_tableau** = [1, 0, 5, **8**, **10**, 20, 52]
5. Etc.

NOTIONS DE BASE DE JAVA

Exercice d'application N°7d

Ecrivez une **fonction** qui **prend en paramètre une chaine de caractère** et **compte le nombre de mots** dans cette chaine.

Exemple : ma_phrase = "Ceci est un exemple de phrase";

f(ma_phrase) = 6

JAVA ORIENTÉ OBJET

JAVA ORIENTÉ OBJET

Notions d'objet

- **Un objet =** représentation symbolique qui a des **caractéristiques** et exécute des **actions**

Exemple d'objets :

Une voiture :

✓ *Caractériques* : marque, modèle, couleur, kilomètres, etc.

✓ *Actions* : rouler, freiner, réparer, etc.

- **Un objet en Java** = instance (ou représentation) d'une ***classe***

JAVA ORIENTÉ OBJET

Notions de classe en Java

- **Une classe =** représentation en Java d'un **ensemble d'objets.** Permet de **créer des objets**

Déclaration d'une classe en Java :

```
class Voiture {
    // Déclaration des caractéristiques (on les appelle attributs en Java)
    String marque;
    String modele;
    String couleur;
    double kilometre;
    // Déclaration des actions (on les appelle méthode en Java)
    void rouler(double kilometre)
     {
         this.kilometre = kilometre;
     }

    }
```

JAVA ORIENTÉ OBJET

Encapsulation

- **public** = les objets ou les méthodes sont accessibles partout dans le projet
- **private** = les objets ou les méthodes ne sont accessibles que dans la classe
- **protected** = les objets ou les méthodes ne sont accessibles que dans la classe ainsi que les sous-classes

	public	protected	défaut	private
Dans la même classe	Oui	Oui	Oui	Oui
Dans une classe du même package	Oui	Oui	Oui	Non
Dans une sous-classe d'un autre package	Oui	Oui	Non	Non
Dans une classe quelconque d'un autre package	Oui	Non	Non	Non

JAVA ORIENTÉ OBJET

Encapsulation

- **Déclaration des encapsulations :**

```
class Voiture {
    // Déclaration des attributs
    private String marque;
    private String modele;
    public String couleur;
    public double kilometre;

    // Déclaration d'une méthode
    public void rouler(double kilometre)
     {
         this.kilometre = kilometre;
     }
    }
```

JAVA ORIENTÉ OBJET

Les constructeurs

- **Un constructeur** = action (méthode) qui permet de "construire" un objet
- **Un constructeur n'est pas obligatoire** (un constructeur par défaut est toujours appelé)
- Une classe peut avoir **plusieurs constructeurs**
- Un constructeur **doit avoir le même nom que la classe sans type de retour**

Déclaration d'un constructeur en Java :

```
class Voiture {
    // Déclaration des attributs
    private String marque;
    private String modele;
    public String couleur;
    public double kilometre;
    // Déclaration d'un constructeur sans paramètre : Même nom que la classe
    public Voiture ()
    {
    }
    // Déclaration d'un constructeur : même nom que la classe
    public Voiture (String marque, String modele, String couleur, double kilometre)
    {
        this.marque = marque;
        this.modele = modele;
        this.couleur = couleur;
        this.kilometre = kilometre;
    }
}
```

JAVA ORIENTÉ OBJET

Les constructeurs de copie

- **Un constructeur qui "copie"** les données d'un objet vers un objet

Déclaration d'un constructeur de copie en Java :

```
class Voiture {
    // Déclaration des attributs
    private String marque;
    private String modele;
    public String couleur;
    public double kilometre;
    // Déclaration d'un constructeur de copie
    public Voiture (Voiture voiture)
    {
        this.marque = voiture.marque;
        this.modele = voiture.modele;
        this.couleur = voiture.couleur;
        this.kilometre = voiture.kilometre;
    }
    }
```

JAVA ORIENTÉ OBJET

La variable this

- **this = l'instance courante de la classe :**
  - ✓ **Accéder au méthode de la classe courante**
  - ✓ **Différencie une propriété d'un objet d'une variable**

Exemple d'utilisation de this :

```
class Voiture {
    // Déclaration d'un constructeur avec paramètre : Même nom que la classe
    public Voiture (String marque, String modele, String couleur, double kilometre)
    {
        this.marque = marque;
        this.modele = modele;
        this.couleur = couleur;
        this.kilometre = kilometre;
        // Appel d'une méthode
        this.uneMethode();
    }
    // Déclaration d'un constructeur sans paramètre : Même nom que la classe
    public void uneMethode ()
    {
    }
}
```

JAVA ORIENTÉ OBJET

Les objets en Java

- **Pour créer un objet, on appelle le constructeur :**
  - ✓ **Appel du constructeur sans paramètre :**

```
Voiture peugeot = new Voiture();
```

  - ✓ **Appel du constructeur avec paramètres :**

```
// Déclaration des attributs
String marque = "Peugeot";
String modele = "3008";
String couleur = "rouge";
double kilometre = 150.2;

// Création d'un objet de voiture le constructeur
Voiture peugeot = new Voiture(marque, modele, couleur, kilometre);
// Création d'un objet par le constructeur de copie
Voiture peugeot_cope = new Voiture (peugeot);
```

JAVA ORIENTÉ OBJET

Exercice d'application N°8

Ecrivez une **classe Eleve avec les informations suivantes** :

- ✓ nom, prenom, age, cycle, moyenne
- ✓ **Créer un objet de la classe Eleve** correspondant à chacun de vous.

JAVA ORIENTÉ OBJET

Les méthodes

- **Une méthode** = action dans une classe

Déclaration d'une méthode en Java :

```
class Voiture {

    // Déclaration d'une méthode
    void rouler(double kilometre)
     {
         this.kilometre = kilometre;
     }

    }
```

Appeler une méthode :

```
MaClasse mon_objet = new MaClasse();

double kilometre = 150;
mon_objet.rouler(kilometre);
```

JAVA ORIENTÉ OBJET

Exercice d'application N°9

Depuis la classe Eleve créée précédemment :

- Écrire une méthode ***afficherInformations()*** pour afficher les informations de l'élève : nom, prenom, age, …

Depuis l'objet eleve créé précédemment :

✓ Appeler la méthode ***afficherInformations*** pour afficher les infos.

JAVA ORIENTÉ OBJET

Les variables static

- **static** : variable commune à tous les objets d'une classe :
  Par exemple : une **constante** ou un **compteur du nombre d'objets d'une classe**
- Accessible dans toute la classe et dans une classe du même package
- Accessible directement depuis la classe

Déclaration et appel d'une variable static à l'intérieur de la classe :

```
class Voiture {
    // Déclaration d'un attribut static
    public static int COUNTER = 0;

    public void compteurVoiture()
     {
         COUNTER++;
     }

    }
```

Appel d'une variable static en dehors de la classe

```
int compteur_static = Voiture.COUNTER;
```

JAVA ORIENTÉ OBJET

Les énumérations

- Permet de contenir une série de données constantes ayant un type sûr
- Se déclare comme une classe, mais avec le mot clé **enum**

Déclaration d'une énumération :

```
public enum Marque {
    PEUGEOT, // Par convention, les éléments sont en Majuscule
    RENAULT,
    MERCEDEZ,
    TOYOTA,
    HUYNDAI; // Le ; met fin à la liste des énumérations
    }
```

Appel d'une variable d'énumération :

```
  public static void main(String args[]){
      for(Marque marque : Marque.values()){
        if(Marque.RENAULT.equals(marque))
          System.out.println("J'aime la marque : " + marque);
        else
           System.out.println(marque);
      }
  }
```

JAVA ORIENTÉ OBJET

Les énumérations : les constructeurs

Déclaration d'une énumération par un constructeur :

```
public enum Marque {
     // Les objets sont directement construits
     PEUGEOT("Marque Peugeot", "France"),
     MERCEDEZ("Marque Mercedez", "Allemagne"),
     TOYOTA("Marque Toyota", "Japon"),
     HYUNDAI("Marque Hyundai", "Corée du Sud");
     // Les paramètres doivent toujours être private
     private String nom;
     private String pays;
     // Constructeur de l'énumérateur
     Marque(String nom, String pays)
     {
          this.nom = nom;
          this.pays = pays;
     }
     // Getter du paramètre pays
     public String getPays()
     {
          return this.pays;
     }
     }
```

Appel d'une variable d'énumération :

```
Marque toyota = Marque.TOYOTA;
System.out.println("J'aime la marque : " + toyota.getPays());
```

JAVA ORIENTÉ OBJET

Bonne pratique de programmation en Java

- Il est **préférable** de **documenter régulièrement son code**
- Une **classe** commence toujours par une **lettre Majuscule**
- Une **propriété** commence par une **lettre minuscule**
- Une **méthode** commence par une **lettre minuscule** et **doit être explicite** : *exemple :* afficherInformation
- Les **paramètres d'une méthode** sont en **minuscule**
- Un **objet** commence par une **lettre minuscule**
- Les **constants et les éléments des énums** doivent être en **Majuscule, …**

- **Pour plus de normes :** https://www.jmdoudoux.fr/java/dej/chap-normes-dev.htm#normes-dev-3

```
class Voiture {
    // Déclaration d'une méthode : commence par une lettre minuscule et les autres mots en
    Majuscule
    void afficherKilometrage(double kilometre)
     {
         System.out.println("Le kilométrage de la voiture est de : " + kilometre);
    }

    }
    Voiture ma_voiture = new Voiture();
```

JAVA ORIENTÉ OBJET

Bon à savoir sur Eclipse …

- Écrire des codes **"snipet"** (ou templates) sur Eclipse fait gagner du temps lors la programmation
- Dupliquez des lignes par **Ctrl + D** (paramétrable depuis **Windows → Préférences → Windows → Keys**)
- Pour les instructions ***if*** et ***for***, lorsqu'il y a **une seule instruction**, **"{"** n'est pas obligatoire
- Lors des affectations, on peut écrire la condition ***if*** sur une seule ligne avec **?** et **:**
- **Évitez**

- **Exemples**

```
// est_positif est true si kilometre est positif et false sinon.
boolean est_positif;
if (kilometre > 0)
     est_positif = true;
 else
     est_positif = false;

// Equivaut aux 5 lignes écrites précédemment : ? = true et : = false
boolean est_positif = (kilometre > 0) ? true : false ;
```

JAVA ORIENTÉ OBJET

Exercice d'application N°10

Écrivez une **classe Rectange avec les attributs suivants** :

✓ **Longueur** et **Largeur**

Écrivez les méthodes suivantes:

- **calculerPerimetre** qui calcule le **périmètre du rectangle**
- **calculerSurface** qui calcule la **surface du rectangle**
- **estCarre** qui **vérifie si le rectange est un carré**
- **toString** qui **affiche** les **attributs du rectange**

Dans la fonction main :

✓ **Créer un tableau de 3 objets de la classe Rectangle.**

✓ Calculez les périmètres et les surfaces des 3 rectangles

✓ Affichez la **somme des périmètres** des rectangles du tableau

JAVA ORIENTÉ OBJET

Exercice d'application N°11

Écrivez une **classe Point** permettant de représenter des points dans **un espace cartésien** avec les attributs suivants :

✓ x et y comme coordonnées du point

- Écrivez une méthode **calculerDistance** qui calcule la **distance entre deux points P1 et P2**
- Écrivez une méthode **calculerOppose** qui retourne **l'opposé d'un point** dans un espace cartésien : **opp(x) = -x et opp(y) = -y**
- Écrivez une méthode **toString**() qui retourne les coordonnées (x,y) d'un point

✓ **Créer 2 objets point1 et point2 de la classe Point.**

✓ Affichez les coordonnées du point **point2**

✓ **Calculez la distance** entre ces deux points

JAVA ORIENTÉ OBJET

Exercice d'application N°12

Écrivez une **classe Livre** permettant de représenter des livres d'une bibliothèque avec les attributs suivants :

- ✓ id, titre, code_isbn, auteur, prix, date_edition
- ✓ **id** doit être **auto-incrément**
- Créer une variable **static compteur** permettant de compter les livres de la bibliothèque
- Écrivez une méthode **toString**() qui retourne les informations d'un livre : Livre N°**id:** Titre : **titre,** Auteur : **auteur,** Prix : **prix,** Date d'édition : **date_edition**
- Les codes ISBN sont **une liste d'énumération :** ISBN_1234, ISBN_4567 et ISBN_7890
- ✓ **Créer un tableau de 3 livres avec chacun un des codes ISBN.**
- ✓ Affichez les informations des livres
- ✓ Affichez **le nombre de livres** de la bibliothèque en utilisant la variable **static compteur**

JAVA ORIENTÉ OBJET

Accessibilité des attributs

- Attributs **doivent être *private*** et accessibles par des méthodes d'accès (ou ***getters***)
- Les attributs sont modifiés par des méthodes (ou ***setters***).
- Les méthodes peuvent être ***public***

Déclaration et appel d'une variable static à l'intérieur de la classe :

```
class Voiture {
      // Déclaration des attributs
      private String marque;
      private String modele;
      private String couleur;
      private double kilometre;
      // Getter de l'attribut « marque »
      public String getMarque()
       {
            return this.marque;
       }
      // Setter (ou modificateur) de l'attribut « marque »
      public void setMarque(String marque)
       {
            this.marque = marque;
       }
      }
```

Accès à la variable « marque » d'un objet voiture en dehors de la classe

```
string marque = voiture.getMarque();
```

JAVA ORIENTÉ OBJET

Exercice d'application N°13

Reprenez **l'exercice N°11** sur la classe **Point :**

✓ Ajoutez l'accesseur **private** aux attributs de la classe

- Créer les getters et les setters de toutes les variables
- Ajoutez une méthode **deplacer** qui permet de déplacer un point de sa position initiale à sa **projection par rapport à l'axe des x**

Dans la fonction main :

✓ Affichez les coordonnées du Point P2 en utilisant les méthodes **getters**

✓ Appelez la fonction **deplacer** pour déplacer le point P2.

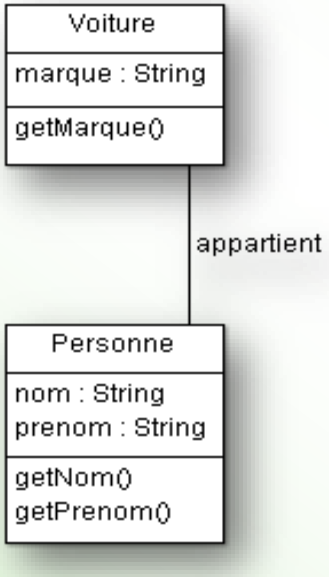
✓ Affichez les **nouvelles coordonnées** du point P2.

✓ Calculer la **nouvelle distance** entre les points P1 et P2.

JAVA ORIENTÉ OBJET

Association des classes

- Une classe peut être appelée dans une ou plusieurs autres classes
- Les liens des classes sont modélisés par le langage UML (Unified Modeling Langage) : exemple de logiciel : **ArgoUML**



JAVA ORIENTÉ OBJET

Association des classes

Déclaration d'une classe dans une autre :

```
class Voiture {
            // Déclaration des attributs
            private String marque;
            private Personne proprietaire;
            // Getter de l'attribut « proprietaire »
            public Personne getProprietaire()
             {
                        return this.proprietaire;
             }
            // Setter (ou modificateur) de l'attribut « proprietaire »
            public void setProprietaire(Personne proprietaire)
             {
                        this.proprietaire = proprietaire;
             }
      }
class Personne {
            // Déclaration des attributs
            private String nom;
            private String prenom;
            // Constructeur par défaut
            public Personne()
            {
            }
      }
```

JAVA ORIENTÉ OBJET

Collections des objets

- Permet de gérer les collections à taille variable
- **Le package java.util.ArrayList** est necessaire pour utiliser les collections

Déclaration d'une collection :

```
ArrayList liste = new ArrayList<T>(); // T étant le type de l'objet de collections
```

Ajout d'un élément dans la collection :

```
liste.add(Objet);
```

Supprime un élément dans la collection :

```
liste.remove(Objet);
```

Retourne un élément dans la collection à l'index i :

```
liste.get(i);
```

Retourne la taille de la collection :

```
liste.size();
```

JAVA ORIENTÉ OBJET

Exercice d'application N°14

Reprenez **l'exercice N°9** sur la classe **Eleve : un élève peut avoir un cycle**

Modélisez ce lien à l'aide d'ArgoUML

Créez une **classe Cycle** avec les attributs suivants :

✓ id, nom, code (id est **auto-incrément**)

- Créer les getters et les setters de toutes les variables

Ajoutez un **attribut Cycle** dans la classe Eleve et lui ajouter des méthodes d'accès

- Renommez la méthode **afficherInformations()** en **toString() et modifiez la** pour qu'elle affiche en plus la filière de l'elève

Dans la fontion main :

✓ Créez une collection de 3 cycles Licence (code L1), Master1 (code M1) et Master2 (code M2)

✓ Créez une collection de 4 autres élèves et leur **ajouter les différents cycles : eleve1 et eleve2 en L1, vous et eleve3 en M1 et enfin eleve4 et eleve5 en M2.**

✓ Affichez les informations de tous les élèves

✓ Affichez la **liste des élèves par cycle**

JAVA ORIENTÉ OBJET

Exercice d'application N°15

Reprenez **l'exercice N°9** sur la classe **Eleve : un élève peut avoir un ou plusieurs cours** et **un professeur peut enseigner un cours**

Modélisez ces liens à l'aide d'ArgoUML

Créez une **classe Cours** avec les attributs suivants :

✓ id, intitule, code (id est **auto-incrément**)

Créez une classe **Professeur** avec les attributs suivants :

✓ titre, nom, prenom, matricule, anciennete. (**titre = Mr.** Ou **Mme**)

- Créer les getters et les setters de toutes les variables des deux classes

Ajoutez un **attribut Cours** dans la **classe Eleve** et un **attribut Professeur** dans la **classe Cours** et leur ajouter des méthodes d'accès

- Ajoutez la méthode **toString() dans les classes Professeur et Cours** pour afficher les informations de chacune des classes

JAVA ORIENTÉ OBJET

Exercice d'application N°15

❖ **Écrivez les différents objets correspondant à cette situation :**

- ✓ Mr. Xavier DUPONT(Mtle : 2018C1, anciennete = 10 ans) enseigne l'Informatique (code cours : INFO)
- ✓ Mme. Celine BERGER (Mtle : 2016B3, anciennete = 6 ans) enseigne la Géomatique (code cours : GEOM)
- ✓ Mr. Sanou PETIT (Mtle : 2000X6, anciennete = 3 ans) enseigne le Géomarketing (code cours : GEOK)
- ✓ Les élèves eleve1 et eleve2 suivent le cours de Géomatique.
- ✓ Vous et les autres élèves suivez le cours d'Informatique

❖ **Affichez les informations des Professeurs et des cours**

❖ **Affichez la liste des élèves par cours**

Changement de situation :

- ✓ Après une année, leurs anciennetés **augmentent d'une année**
- ✓ Mr. Xavier DUPONT enseigne maintenant l'Électronique (code cours : ELEC)

❖ **Modifiez les différents objets correspondant à cette nouvelle situation**

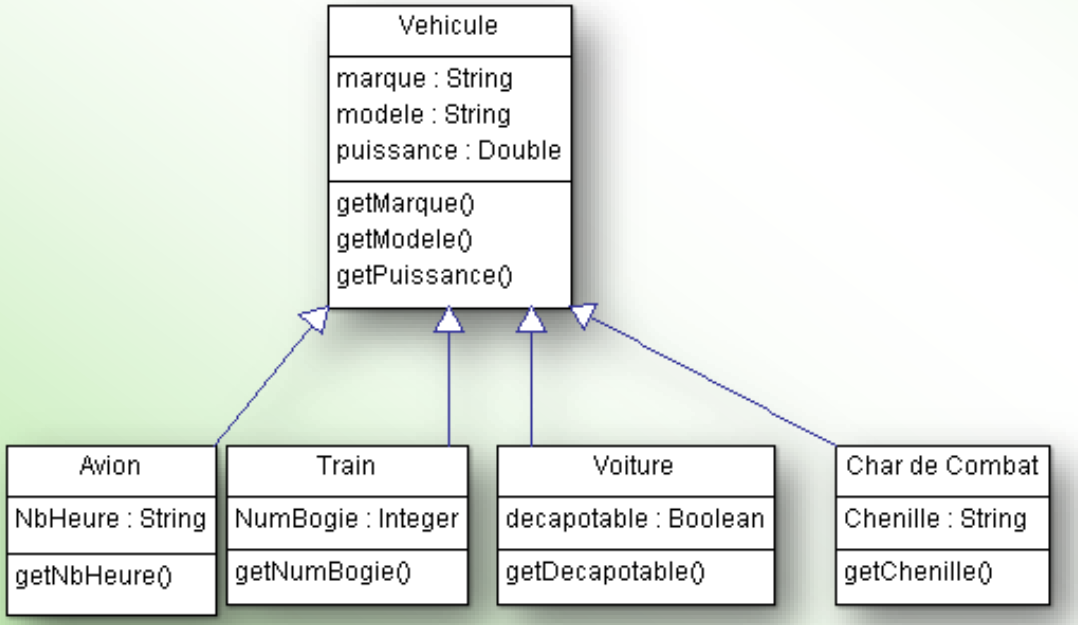
❖ **Affichez les informations des professeurs et des cours**

JAVA ORIENTÉ OBJET

Héritage

- Une classe peut hériter d'une autre : la classe héritée est la ***classe mère*** et celle qui hérite devient ***classe fille***
- La **classe fille hérite de tous les attributs et méthodes de la classe mère**
- Une **classe mère peut posséder une ou plusieurs classes filles**
- Une **classe fille ne peut hériter que d'une seule mère**

- **Modélisation avec ArgoUML :**



JAVA ORIENTÉ OBJET

Héritage : exemple d'un héritage

```
class Vehicule { // Classe mère
            private String marque;
            private String modele;
            private double kilometre;
            // Constructeur de la classe mère
            public Vehicule(String marque, String modele, double kilometre)
            {
            }
            public void toString() {
            }
}
class Voiture extends Vehicule { // Classe fille
            // Déclaration des attributs propres à « Voiture »
            private bool decapotable;
            // Constructeur de la classe « Voiture »
            public Voiture(String marque, String modele, double kilometre, bool decapotable)
            {
                  // Fait appel au constructeur de la classe mère
                  super(marque, modele, kilometre);
                  this.decapotable = decapotable;
            }
      }
class Train extends Vehicule { // Classe fille
            // Déclaration des attributs propres à « Train »
            private String bogie;
            // Constructeur de la classe « Train »
            public Train()
            {
            }
}
```

JAVA ORIENTÉ OBJET

Exercice d'application N°16

On vous a chargé de développer un logiciel de gestion des utilisateurs de l'entreprise ESN-IT.

1. Effectuez le diagramme de classe avec ArgoUML sachant que :

- Un développeur est caractérisé par : id, nom, prenom, mail, telephone, salaire, specialite
- Un Manager est caractérisé par : id, nom, prenom, mail, telephone, salaire, code_manager
- Une assistante de direction est caractérisé par : id, nom, prenom, mail, telephone, salaire, code_assistance
- Chaque salarié est rattaché à un service caractérisé par : id, numéro, telephone
- Le manager est manager d'un ou plusieurs développeurs

NB : Tous les identifiants sont auto-incréments

JAVA ORIENTÉ OBJET

Exercice d'application N°16

Dans la méthode ***main*** :

1. Créez 2 services : Informatique (N° : IN10) et DRH (N° : DR45)
2. Créez 2 développeurs du service Informatique avec respectivement comme spécialités Java et PHP dont l'un touche 40000k et l'autre 30000k
3. Créez 1 manager du service Informatique qui touche 80000k (code manager : MNI)
4. Créez 1 assistante de direction du service DRH touchant 40000k (code assistance : ASR)
5. Le manager dirige l'équipe des développeurs
6. Affichez les informations du manager, des développeurs et de l'assistante de direction.
7. Affichez les développeurs managés par le manager touchant entre 40000k et 60000k
8. Le manager a eu une augmentation de 20% par rapport à son salaire annuel.
9. Calculez le nouveau salaire du manager et affichez-le.

JAVA ORIENTÉ OBJET

Héritage : notions de redéfinition

- **Signature d'une méthode : nom** de la méthode, **type** et **nombre** de paramètres

  Exemple: ces 3 méthodes ont **des signatures différentes**

```
public int somme(int a, int b) {
}
public int somme(int a, int b, int c) {
}
public double somme(double a, double b) {
}
```

- Une méthode d'une classe fille peut redéfinir une ou plusieurs méthodes de sa classe mère **si elles ont la même signature** :

  on dit que la méthode de la ***classe fille redéfinit celle de la classe mère***

JAVA ORIENTÉ OBJET

Héritage : notions de redéfinition

```
class Vehicule { // Classe mère
            private String marque;
            private String modele;
            private double kilometre;
            // Constructeur de la classe mère
            public Vehicule(String marque, String modele, double kilometre)
            {
            }
            public void toString() {
                System.out.println("Affichage des informations d'un Véhicule : ");
            }
}
class Voiture extends Vehicule { // Classe fille
            // Déclaration des attributs propres à « Voiture »
            private bool decapotable;
            // Constructeur de la classe « Voiture »
            public Voiture(String marque, String modele, double kilometre, bool decapotable)
            {
                 // Fait appel au constructeur de la classe mère
                 super(marque, modele, kilometre);
                 this.decapotable = decapotable;
            }
            public void toString() {
                super.toString(); // Appelle la méthode toString() de la classe mère
                System.out.println("Une voiture est un véhicule qui a ces informations en plus :");
                System.out.println("Décapotable :" + this.decapotable);
            }
     }
```

JAVA ORIENTÉ OBJET

Exercice d'application N°17

- Ajoutez une **classe Helicoptere qui hérite de Vehicule** avec les attributs suivants :
  - ✓ nombre_helices, rotor
- Redéfinissez la méthode **toString() dans la classe Helicoptere**
- Ajoutez une méthode ***rouler(double kilometre)*** dans la **classe Vehicule** qui **ajoute un kilométrage au véhicule**
- Redéfinissez cette méthode **dans toutes les classes filles** avec la particularité suivante :
  - En plus du ***kilometre***, on **ajoute à un hélicoptère 5kms** de plus correspondant à sa distance avant atterrissage.

Dans la classe main :

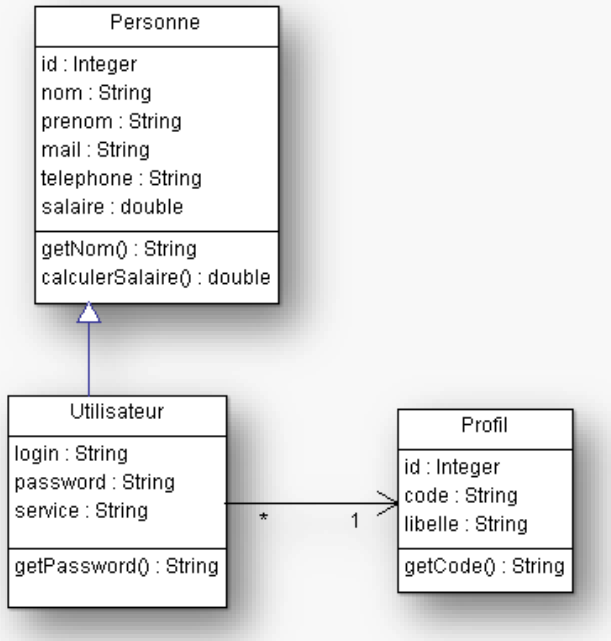
- Créez **2 voitures** et **1 hélicoptère**
- **Faites rouler tous les objets de 50kms**
- Affichez les informations des voitures et de l'hélicoptère grâce à la méthode toString()

JAVA ORIENTÉ OBJET

Exercice d'application N°18

Le directeur des systèmes d'information de la société ESN-IT souhaite développer un module pour la gestion des utilisateurs de son service informatique, pour cela il vous a fait appel pour réaliser cette tâche.

Le diagramme de classe a été établi par un analyste :



JAVA ORIENTÉ OBJET

Exercice d'application N°18

1. Développer les classes ci-dessus en JAVA dans un package de votre choix (**id est auto-incrément**)
2. **Redéfinir** la méthode **calculerSalaire()** et la méthode **toString**() dans la classe Utilisateur.

   Sachant que :

   - Le Manager aura une augmentation de 10% par rapport à son salaire normal,
   - Le Directeur Général aura une augmentation de 40% par rapport à son salaire normal.
3. Créer les profils :
   - Chef de projet (code : CP),
   - Manager (code : MN),
   - Directeur de projet (code : DP),
   - Directeur des ressources humaines (code : DRH),
   - Directeur Général (code : DG)
4. Créer des utilisateurs avec les différents profils métiers.
5. Afficher la liste des utilisateurs.
6. Filtrer la liste et afficher la liste des managers.

JAVA ORIENTÉ OBJET

Polymorphisme

- La faculté attribuée à un **objet** d'être **une instance de plusieurs classes**
- **Un objet d'une classe fille peut être créé à partir d'une classe mère**
- **Avantages → création d'ensembles regroupant des objets de classes différentes**

```
Vehicule[] vehicules = new Vehicule[3];
vehicules[0] = new Voiture("Peugeot", "407", 100.50);
vehicules[1] = new Voiture("Renault", "Clio4", 500);
vehicules[2] = new Train("Alstom", "A10", 50000);
```

- **On affiche les types des objets :**

```
for(int i = 0; i < vehicules.lenght; i++)
{
   if(vehicules[i] instanceof Vehicule)
      System.out.println("élément :" + i + " est un Véhicule.");

   if(vehicules[i] instanceof Voiture)
      System.out.println("élément :" + i + " est une Voiture.");

   if(vehicules[i] instanceof Train)
      System.out.println("élément :" + i + " est un Train.");
}
```

JAVA ORIENTÉ OBJET

Exercice d'application

- Depuis l'exemple du cours, ajoutez 2 nouveaux hélicoptères dans le tableau **vehicules**
- Affichez les **informations de tous les véhicules créés** grâce à la fonction toString()

JAVA ORIENTÉ OBJET

Exercice d'application N°19

Reprenez **l'exercice N°17** sur l'entreprise ESN-IT :

Dans la fonction main :

1. Créez un tableau de 5 personnes
2. Ajoutez les 2 développeurs, le manager et l'assistante de direction dans ce tableau
3. Il y a eu 2 nouveaux embauchés dans l'entreprise :
    - Ajoutez 1 développeur comme spécialité Géomatique au service Informatique qui touche 25000k
    - Ajoutez son manager (code MN-G5) du même service qui touche 60000k
4. Affichez les informations des différents salariés de l'entreprise en parcourant le tableau des personnes.

JAVA ORIENTÉ OBJET

Classe abstraite

- Une classe abstraite est une **classe mère qui ne peut être instanciée**

```
Vehicule vehicule = new Vehicule(); // Pas de constructeur
```

- Ces classes doivent **obligatoirement être héritées**
- Les méthodes sont **redéfinies dans les classes filles**
- Elle **peut contenir une ou plusieurs méthode abstraite**
- Toutes les classes filles **doivent redéfinir toutes les méthodes abstraites de la classe abstraire**

```
public abstract class Vehicule {
        // Déclaration des attributs
        public String marque;
        public Personne proprietaire;

        // Affiche les informations d'un véhicule
        public void toString()
        {
            System.out.println(« Ceci est un Véhicule.");
        }
        // Une méthode abstraite ne doit rien contenir (que sa signature) et doit être redéfinie
        // Pas de { }
        public abstract void rouler();
    }
```

JAVA ORIENTÉ OBJET

Exercice d'application N°20

Reprenez **l'exercice N°17** sur l'entreprise ESN-IT :

1. Rendez la **classe mère Personne abstraite**
2. Ajoutez un **attribut anciennete à la classe Personne** qui détermine l'ancienneté d'un salarié
3. Ajoutez une **méthode abstraite ajouterPrimes** qui permet d'ajouter une prime à un salarié sachant que dans cette entreprise est fonction de l'ancienneté du salarié et du CA de l'entreprise :

   prime = salaire*anciennete*20% (voir le calcul de la prime)

4. Rendez **toutes les méthodes de la classe abstraites**

Dans la fonction main :

5. Affichez les informations des différents salariés
6. L'entreprise a réalisé un CA de 150000k € cette année. Calculez les primes des différents salariés sachant que :
   - Le développeur Java a une ancienneté de 5 ans
   - Le développeur PHP : 2 ans

Exercice d'application N°20

- Le développeur Géomatique : 1 ans
- Le manager : 10 ans
- L'assistante de direction : 4 ans

6. Affichez les primes de chacun des salariés et leurs salaires respectifs à la fin du mois.

Attention : **Leurs salaires sont donnés en annuels bruts !**

JAVA ORIENTÉ OBJET

Exercice d'application N°21

Vous avez été nommé Chef de projet Informatique au Chef d'Etat Major des Armées.

On vous demande de développer un logiciel permettant la gestion des différentes armées Françaises.

Pour informations :

1. L'armée Française comprend 3 grandes armées : l'Armée de Terre, la Marine Nationale et l'Armée de l'Air
2. Chaque Armée (id, code) dispose d'un budget annuel et d'un Général
3. Chaque armée dispose d'un certain nombre de personnels militaires et civils
4. L'Armée de Terre dispose des militaires repartis en : l'infanterie, l'armée blindée et l'artillerie
5. L'Armée de Terre dispose également de matériels de combat : blindées, camions, chars de combat, hélicoptères, système sol-air et d'avions
6. Les militaires peuvent être équipés : gilet par balles, casque, fusils d'assaut
7. Les militaires appartiennent à un statut donné : Sous-Officier, Officier, Lieutenant, Colonel, Général, Maréchal
8. Les civils appartiennent à un métier particulier : Informaticien, Mécanicien, Électronicien, Électricien
9. Les militaires peuvent être envoyés au combat dans un pays donné
10. La Marine Nationale, elle, dispose de : sous-marins, sous-marin nucléaire et d'un porte-avion

JAVA ORIENTÉ OBJET

Exercice d'application N°21

Travail à faire :

1. Modélisez votre armée à l'aide d'Argo-UML à partir des informations fournies
2. Créez 3 militaires :
    1. Général de l'armée de terre Arnaud B.
    2. Lieutenant Sami BENNANI
    3. S-Officier Thomas ZIDANE
3. Créez l'armée de terre sachant qu'elle possède un budget de 2 Milliards d'€/an
4. Créez 2 civils : un électricien et un mécanicien
5. Créez un véhicule blindé, un camion et un char de combat
6. Thomas ZIDANE possède une mitrailleuse de type MP5

JAVA ORIENTÉ OBJET

Les interfaces

- Objectif => utiliser les objets dans un autre type d'application (réutilisabilité)
- ***Exemple*** : un garagiste (comme Speedy) vous demande de réutiliser vos objets **Vehicule** pour sa nouvelle application

1. On met la méthode ***remplacerPneus()*** dans la classe mère **Vehicule** :

```
public abstract class Vehicule {
        // Déclaration des attributs
        public String marque;
        public Personne proprietaire;

        // Remplace les pneus d'un véhicule
        public void remplacerPneus()
        {
            System.out.println("Ici on répare les pneus d'un Véhicule.");
        }
        // Une méthode abstraite ne doit rien contenir (que sa signature) et doit être redéfinie
        // Pas de { }
        public abstract void rouler();
    }
```

Le problème, c'est qu'un train ne peut avoir de pneus à remplacer

Les interfaces

2. On peut mettre la méthode **remplacerPneus()** dans la classe Voiture :

```
public class Voiture extends Vehicule {
          // Déclaration des attributs
          private bool decapotable;
          // Remplace les pneus d'une voitures
          public void remplacerPneus()
          {
             System.out.println("Ici on répare les pneus d'un Véhicule.");
          }
     }
```

Le problème, c'est qu'il n y aura plus de polymorphisme => pas d'appel de ces méhodes via un objet Vehicule

```
Vehicule[] vehicules = new Vehicule[2];
vehicules[0] = new Voiture("Peugeot", "407", 100.50);
vehicules[1] = new Voiture("Renault", "Clio4", 500);

// Appel de la méthode remplacerPneus
Vehicules[0].remplacerPneus();
```

Impossible car un objet Vehicule n'a pas la méthode remplacerPneus()

JAVA ORIENTÉ OBJET

Les interfaces

- **Objectif** => utiliser les objets dans un autre type d'application (**réutilisabilité**)
- N'implémente aucune méthode
- Une classe **peut hériter de plusieurs interfaces**
- Des classes **non liées hiérarchiquement peuvent implémenter la même interface**
- Une classe qui implémente une interface **redéfinit toutes ses méthodes**

```
public interface IVoiture {
        // Déclaration des attributs
        public int[] pneus;
        // Remplace les pneus d'un véhicule
        public void remplacerPneus()
        {
        }
}
class Voiture implements Ivoiture { // Classe implémentant l'interface IVoiture
        // Déclaration des attributs
        public boolean decapotable;
        // Remplace les pneus d'un véhicule
        public void remplacerPneus()
        {
           System.out.println("Ici on répare les pneus d'un Véhicule.");
        }
}
```

JAVA ORIENTÉ OBJET

Les interfaces

```
IVoiture[] voitures = new IVoiture[2];
vehicules[0] = new Voiture("Peugeot", "407", 100.50);
vehicules[1] = new Voiture("Renault", "Clio4", 500);

// Appel de la méthode remplacerPneus
Vehicules[0].remplacerPneus();
```

✓OK ➔ une voiture implémente l'interface IVoiture

JAVA ORIENTÉ OBJET

Exercice d'application N°22

- Reprenez **l'exercice N°11** sur la classe Point :

1. Créez une nouvelle classe **Segment** qui est définie par 2 points et une couleur, codée par un entier positif
2. Une classe **Triangle** définie par 3 points et une couleur, codée par un entier positif
3. Créez deux interfaces :
    1. **IColoration** qui permet de colorier nos graphiques (Triangle, Rectangle) : comprend les méthodes **setCouleur(int couleur)**, **getCouleur()** et **draw()**
    2. **IDeplacement** qui permet de déplacer les graphiques : comprend la méthode **translation(int dx, int dy)** qui permet d'obtenir la translation d'un point par un vecteur (dx, dy)
4. La classe **Point** implémente l'interface **IDeplacement** et les classes **Segment** et **Triangle** implémentent **les deux interfaces.**
5. Pour des raisons de simplicité, la méthode **draw() affiche simplement** les informations d'un segment : le point début, le point final et la couleur

JAVA ORIENTÉ OBJET

Exercice d'application N°22

- Dans la méthode main :

1. À partir des deux points P1 et P2 créés dans l'exercice 11, créez 1 segment S1 de couleur rouge (code 2)
2. Créez deux autres segments S2 (vert, code 1) et S3 (bleu, code 3) à partir d'autres points
3. Créez un triangle T1 à partir des segments S1, S2 et S3.
4. Appelez la méthode **translate()** pour effectuer une translation du triangle T1 par le vecteur **(-2, 3)**

1. Appelez la méthode **draw()** du triangle T1 pour afficher les informations de ce dernier

JAVA ORIENTÉ OBJET

Les exceptions

- Une erreur **bloque** le processus d'exécution d'un programme
- Exemple :

```
class Test {
        // Fonction main
        public static void main(args[])
         {
                int a = 10;
                int b = 0;

                int result = a / b;

                System.out.println("Le résultat de la division de " + a + " par " + b + " : " + result);
         }
    }
```

Erreur : Division par 0 impossible ➔ l'exécution du programme s'arrête !!
Donc pas d'affichage du message "Le résultat de la disition de 10 par 0 : "

JAVA ORIENTÉ OBJET

Les exceptions

- Il faut gérer les **exceptions** dans les programmes ➔ récupérer l'erreur éventuelle et la traiter
- Une fois l'exception traitée, **le programme ne bloque pas et continue d'être exécuté**

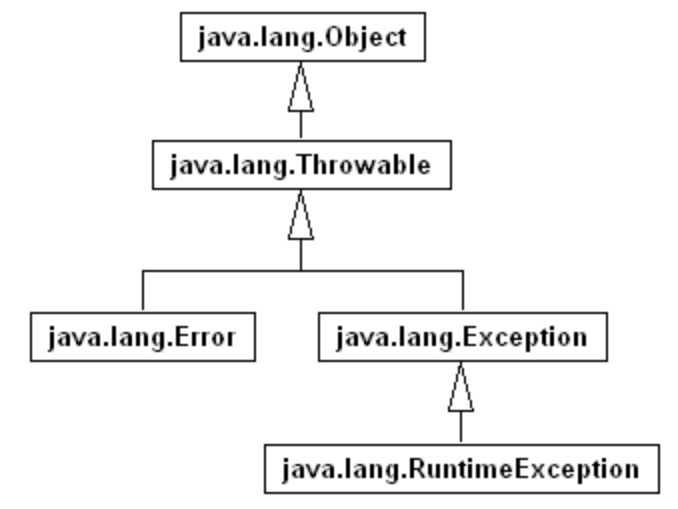
```
try {
        // On exécute le code dans le bloc try
        int a = 10;
        int b = 0;

        int result = a / b;

        System.out.println("Le résultat de la division de " + a + " par " + b + " : " + result);
    }
    catch(ArithmeticException exception) {
        // On "capture" les erreurs dans le bloc catch
        System.out.println("Division par 0 impossible.");
    }

    System.out.println("On est à la fin de l'exécution du programme.");
```

JAVA ORIENTÉ OBJET

Les exceptions personnalisées

- Il existe deux principals sous-classes de la classe ***Throwable***
  - ✓ ***Exception :*** signale une erreur dans l'application
  - ✓ ***Error*** : signale une erreur plus grave au niveau système (manque de ressource, mauvais format de classe, etc.)

- Par convention, les exceptions se terminent par Exception



JAVA ORIENTÉ OBJET

Les exceptions personnalisées

- Création d'une classe d'exception personalisée

```
public class VoitureException extends Exception
 {
     private double kilometre;
     public VoitureException(String message)
     {
          System.out.println("Erreur : " + message);
          this.kilometre = kilometre;
     }
}
```

- L'appel de la classe d'exception

```
double kilometre = -2;
if(kilometre < 0)
    throw VoitureException("Le kilométrage doit être positif");
```

- L'appel de la classe d'exception dans une méthode

```
public double testKilometrage(double kilometre) throws VoitureException
{
         return kilometre;
}
```

JAVA ORIENTÉ OBJET

Exercice d'application N°23

Reprenez **l'exercice N°22** sur l'entreprise ESN-IT :

- Créez une classe personnalisée **EmployeException** d'exception sachant que :
  - ✓ Les salaires des employés sont des nombres positifs
  - ✓ Les codes des employés ne doivent pas dépasser 3 caractères et ces derniers doivent être en Majuscule
  - ✓ L'ancienneté des employés est un nombre entier positif
  - ✓ L'email doit contenir le caractère '@'

- Dans la classe ***main***
  - Utilisez la classe personnalisée pour instancier les différents objets
  - Testez votre exception personnalisée avec des salaires négatifs, un code à 4 lettres et en minuscule et afficher les erreurs
  - Testez l'exception personnalisée sur l'une ancienneté avec un nombre décimal

JAVA ORIENTÉ OBJET

Gestion des flux : lecture/écriture dans un fichier

- Classe File :
- Flux de lecture : **Reader** (Classe permettant la gestion des flux de lecture)
- Flux d'écriture : **Writer** (Classe permettant la gestion des flux en écriture)
- Avec les fichiers, on utilise les classes **FileReader** et **FileWriter**
- Les méthodes de lecture et d'écriture **déclenchent des exceptions IOException ou FileNotFindException** si **le fichier n'est pas trouvé**

- Lire un fichier monFichier.txt

```
File mon_fichier = new File("mon_fichier.txt");
FileReader out = new FileReader(mon_fichier);

// A la fin du fichier, la méthode read() retourne -1
while((c == out.read()) != -1)
{
   System.out.println("On a lu le caractère : " + c);
}
// On doit toujours fermer le flux
out.close();
```

JAVA ORIENTÉ OBJET

Gestion des flux : lecture/écriture dans un fichier

- Écrire dans un fichier mon_fichier2.txt

```
File mon_fichier2 = new File("mon_fichier2.txt");
FileWriter in = new FileWriter(mon_fichier2);

int c = 'a';

// On écrit dans le fichier mon_fichier2.txt
in.write(c);

// On doit toujours fermer le flux
in.close();
```

JAVA ORIENTÉ OBJET

Exercice d'application N°24

- Créez une classe Copie qui permet de **lire les données** du fichier **test.txt**, de **créer un deuxième fichier** (**test_copie.txt**) et de **copier** les données du fichier test.txt dans test_copie.txt.
- Écrivez les méthodes nécessaires aux différentes opérations
- Gérez les différentes exceptions qui puissent être déclenchées (FileNotFindException, IOException, …)