Cours de Java

Ahmed Zidna, Bureau : B37.
Département Informatique de l'IUT
Univerisité de Lorraine, Ile du Saulcy, F-57045 METZ
ahmed.zidna@univ-lorraine.fr

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

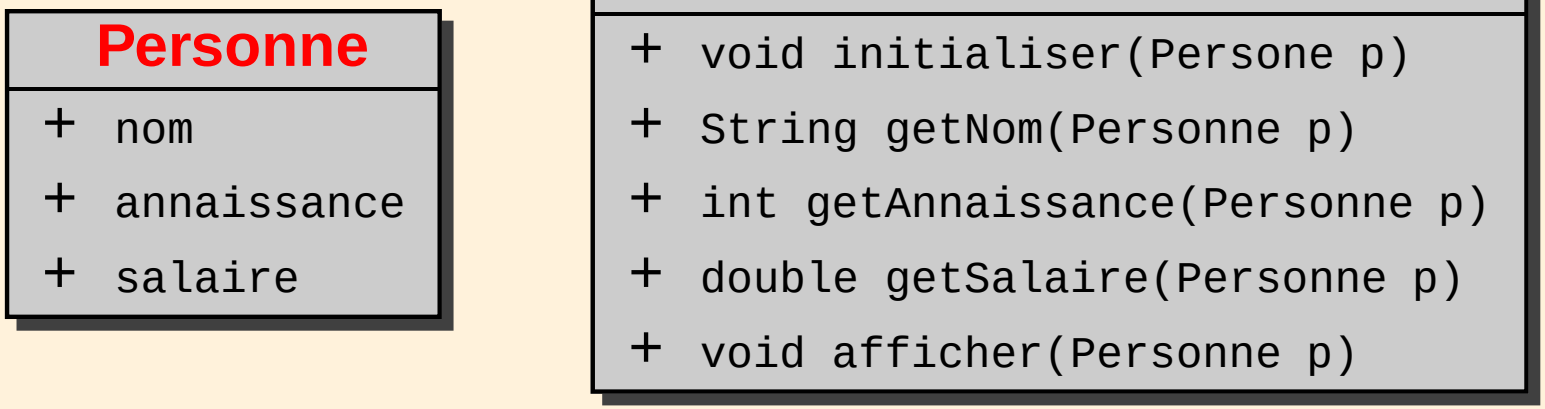
Sommaire

Sommaire

Sommaire

De la structure à la classe I

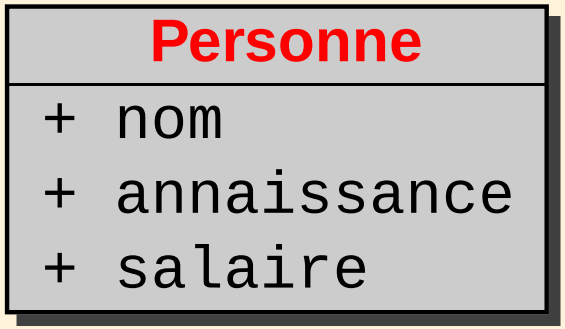
- En programmation classique, il y a séparation complète entre les données et les méthodes.



De la structure à la classe I

- Structure de données Personne en langage C :

```
1  struct Personne{
   string nom;
3  int annee_naissance;
   double salaire;
5  };
```



De la structure à la classe I

- Méthode initialiseer. Elle retourne une variable de type Personne.

```
1  Personne initialiser(string ch,int a,double s){
   Personne p;
3  p.nom=ch; p.annee_naissance=a; p.salaire=s;
   return p;
5  }
```

- Méthode afficher. Elle a un paramétre de type Personne.

```
1  void afficher(Personne p){
   System.out.println(" Personne :"+ getNom(p)+" "+
3  getAnnee(p)+ "  "+getsalaire(p);}
   }
```

De la structure à la classe I

- Voici la primitive d'accès au nom d'une personne

```
String getNom(Personne p){
return p.nom;
}
```

- Voici la primitive d'accès à l'année de naissance d'une personne

```
int getAnnee(Personne p){
return p.annee_naissance;
}
```

- Voici la primitive d'accès au salaire d'une personne

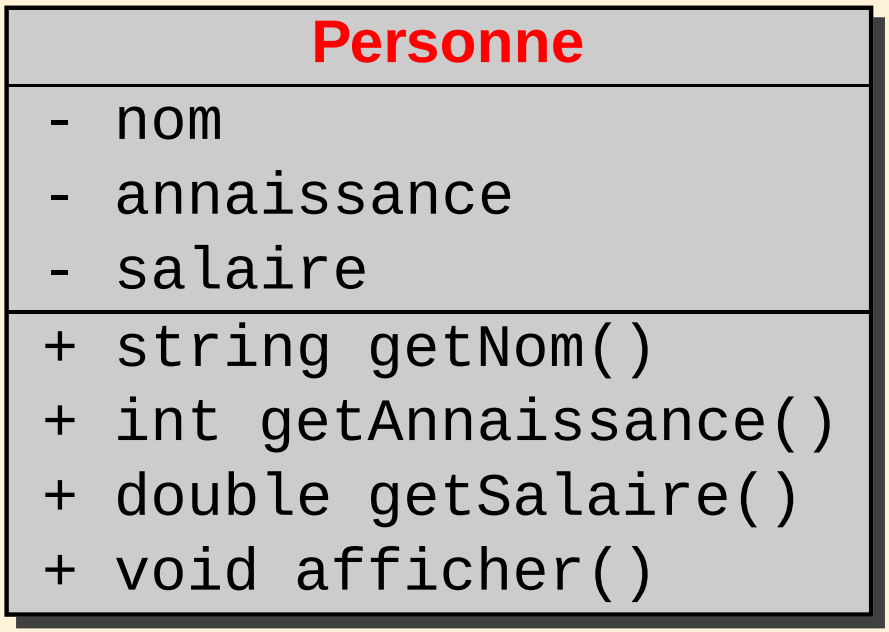
```
double getSalaire(Personne p){
return p.salaire;
}
```

- Voici un programme qui teste Personne

```
1  main()
   {
3  Personne p=initialiser("Dupont",1986,2300);
   afficher(p);
5  }
```

De la structure à la classe I

La classe regroupe les données et les méthodes :



De la structure à la classe I

- Voici la classe Personne

```
1  class Personne{
   //------------------------------
3  // attributs prives
   //------------------------------
5  private string nom;
   private int annee_naissance;
7  private int salaire;
   //------------------------------
9  // methode publiques
   //------------------------------
11 public void initialiser(string ch,int a,int s){.....}
   public string getNom(){.....}
13 public int getAnnee(){.....}
   public int getSalaire(){.......}
15 public void afficher(){.......}
   }
```

De la structure à la classe I

- fonction qui initialise une personne

```
   void initialiser(string ch,int a,int s){
2  nom= ch;
   annee_naissance=a;
4  salaire=s;
   }
```

- Voici les primitives d'accès aux champs d'une personne

```
1  string getNom(){ return nom;}
   int getAnnee(){return annee_naissance;}
3  int getSalaire(){return salaire;}
```

- fonction qui affiche une personne

```
1  void afficher(){
   System.out.println("␣Personne␣:"+ getNom()+"␣␣"+
3  getAnnee()+ "␣␣"+getSalaire());;
    };
```

Voici un programme qui teste Personne I

```
class TesPersonne{
public static void main(string[] args)
{
Personne p.initialiser("Dupont",1986,2300);
p.afficher();
}
```

Classe et objets I

- Le concept clé des LOO est la notion de classe.
- La classe est une entité qui regroupe des structures de données et les opérations associées à ces données :
  1. données membres, ou les attributs de la classe
  2. fonctions membres, ou les méthodes de la classe
- Les instances de la classe sont des objets

Protection des membres d'une classe I

- Les membres peuvent être :
  - **private** : seules les fonctions membres de la classe y ont accès.
  - **public** : toute fonction extérieure à la classe y a accès *(en consultation et en modification)*
  - **protected** : Cette notion est liée à l'héritage.
- L'objet pour lequel une fonction membre est appelée constitue toujours un argument explicite de celle-ci. Il est toujours accessible et un pointeur est toujours associé à cet objet : **this**

Constructeur d'une classe I

- Un constructeur est chargé d'initialiser une instance de la classe au moment de la création de l'objet avec new
- Il porte le nom de la classe
- Une classe peut avoir plusieurs constructeurs :
  - sans paramétre,
  - avec des paramétres,

Classe Personne I

- Voici le constructeur Personne de la classe.

```
   public Personne(string n, int a, double s){
2  setNom(n);
   setAnneeNaissance(a);
4  setSalaire(s);
   }
```

- Voici les primitives d'accès : les getters

```
1  public string getNom(){return nom;}
   public int getAnnee_naissance(){return annee_naissance;}
3  public double getSalaire(){return salaire;}
```

- Voici les primitives de modifications : les setters

```
1  public void setNom(string n){nom=n;}
   public void setAnneNaissance(string a){annee_aissance=a;}
3  public void setSalaire(double s){salaire=s;}
```

Classe Personne II

- Voici la fonction afficher

```
1  public void afficher(){
   System.out.println( nom +"␣"+annee_naissance+"␣"+salaire);
3  }
```

Et pour tester la classe Personne I

```
1  class TesPersonne
   public static void main(string args[])
3  {
   Personne p1=new Personne("David Tonsac", 1962, 2500);
5  Personne p2=new Personne("Andre Sanfrape", 1970, 4500);
   p1.afficher();
7  p2.afficher();
   }
```

Implémentation complète de Personne I

- New alloue la mémoire de l'objet, appelle le constructeur et retourne l'adresse de l'objet construit.
- Deux valeurs retournées : la référence au type Personne e l'adresse de l'objet

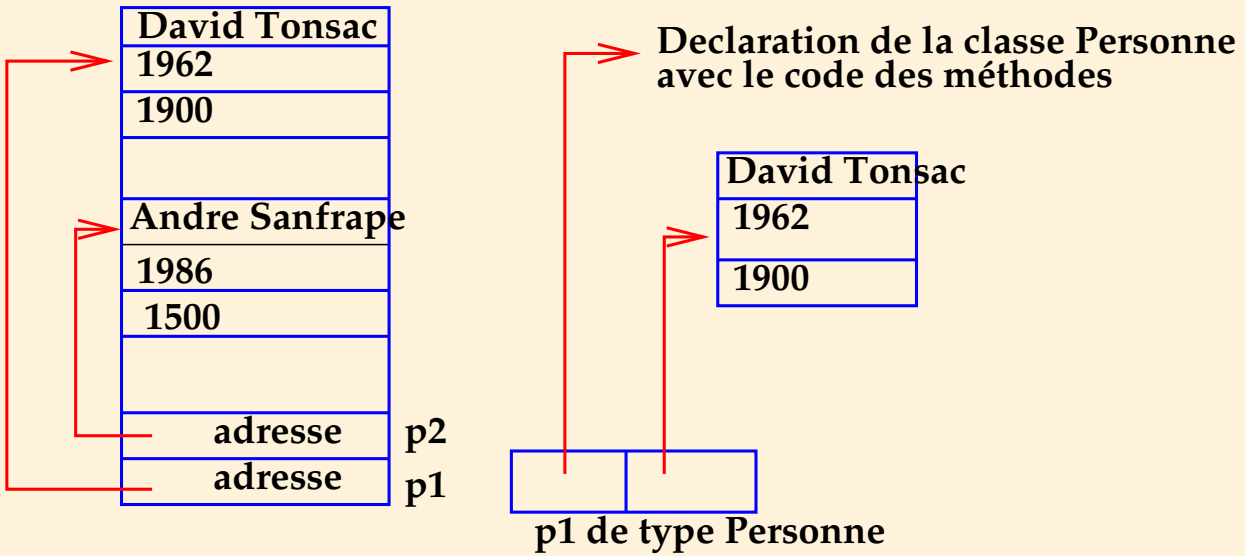


FIGURE: Implémentation de la classe Personne

Surcharge du constructeur Personne I

1 constructeur à 3 paramétres

```
   public Personne(String n , int a , int s){
2  nom=n; annee_naissance=a; salaire=s;
   }
```

2 constructeur à deux paramétres

```
1  public Personne(String n , int a){
   nom=n; annee_naissance=a; salaire=1200;
3  }
```

3 Constructeur de copie

```
1  public Personne(Personne p){
   nom=p.nom; annee_naissance=p.annee_naissance; salaire=p.
       salaire;
3  }
```

Un constructeur sans paramétres I

- On peut crèer un objet de type Personne dont les valeurs sont fournies par l'utilisateur par une saisie au clavier :

```
1  public Personne(){
   Scanner sc = new Scanner(System.in);
3  System.out.print("Donnez le nom: ");
   nom = sc.nextLine();
5  System.out.println("Donnez l'annee de naissance");
   annee_naissance = sc.nextInt();
7  System.out.println("Donnez le salaire");
   salaire = sc.nextDouble();
9  }
```

Voici une classe avec une surcharge I

- Voici la fonction afficher surchargée
  1. sans paramétres

```
1  public void afficher(){
   System.out.println(nom+"␣"+annee_naissance+"␣"+salaire
       );
3  }
```

  2. avec 1 paramétre

```
1  public void afficher(String message)
   {
3  System.out.println(message +"␣"+nom+"␣"+
       annee_naissance+"␣"+salaire);
   }
```

Classe Fraction I

Une fraction est un nombre sous la forme a/b où $a, b \in IN$ avec $b \neq 0$

Fraction
- num - den
+ Fraction(int, int) + Fraction somme(Fraction) + int getNum() + int getDen() + void setNum(int) + void setDen(int) + void afficher()

Classe Fraction I

- Voici la classe : un constructeur et la méthode somme.

```
   class Fraction{
2  private int num;
   private int den;
```

- Voici le constructeur

```
1  public Fraction(int a,int b){
   num=a;
3  den=b;}
```

- Voici la fonction somme des deux fractions this et x

```
1  public Fraction somme(Fraction x){
   return new Fraction(this.num*x.den+this.den*x.num, this.den
       *x.den);
3  }
   }
```

Classe Fraction I

- Voici la classe TestFraction

```
class TestFraction{
public void main(String [] arg){
Fraction x1, x2, x3;
x1=new Fraction(2,3);
x2=new Fraction(1,4);
x3=x1.somme(x2);
}
}
```

Variables de classe : static I

- variable d'instance (private , protected..),
- variable locale (fonctions)
- variable globale : commune à toutes les instances de la classe et appelée variable de classe. Elle est définie avec le mot clé static.

```
class Math{
public static double PI = 3.1416;
.....
}
```

- à l'appel :

```
double x = Math.PI;
```

Méthode de classe : static

Que faire lorsqu'on désire écrire une fonction qui n'est pas membre d'une classe ?

Fonction membre

Une fonction membre d'une classe.
a toujours au moins un argument implicite..

Fonction static

Elle es définie dans une classe, avec le mot static.
A l'appel de la fonction : Nom-Classe.nomDeFonction().

Exemple de fonction non membre d'une classe I

- La méthode **perimetre** de la classe **cercle** est surchargée.

```
1  class Cercle{
   private int rayon;
3  Cercle(int rayon){
   this.rayon=rayon;
5  }
```

- Fonction non membre perimetre

```
1  static double perimetre(Cercle c){
   return 2*Math.PI*c.rayon;
3  }
```

- Fonction membre perimetre

```
1  double perimetre(){
   return 2*Math.PI*rayon;
3  }
```

Exemple de fonction non membre d'une classe II

- Voici comment je dois les appeler

```
1  Cercle c=new Cercle(3);
   double p=Cercle.perimetre(c); //pour la premiere
3  double p=c.perimetre(); //pour la seconde
```

Exemple : fonctions qui calcule le maximum I

```
1 public class Math{
```

- calcule et renvoie le maximum de 2 entiers

```
1 public static int max(int a,int b){
  if (a>=b) return a; else return b;
3 }
```

- calcule et renvoie le maximum de 2 doubles

```
1 public static double max(double a,double b){
  if (a>=b) return a; else return b;
3 }
```

- calcule et renvoie le maximum de 3 entiers

```
1 public static int max(int a,int b,int c){
  return max(a,max(b,c));}}
```

Exemple : fonctions qui calcule le maximum II

- A l'appel :

```
class TestMath{
int n1,n2,n3,n4,n5;
double u,v,w;
n1=3;n2=5;n3=8;
//pas d'argument implicite a l'appel de max(int,int)
n4=Math.max(n1,n2);
n5=Math.max(n1,n2,n3);
u=2.5;v=-3.8;
//appel de la fonction Math.max(double a,double b)
w=Math.max(u,v);
}
```