

Cours .NET avec C#

Programmation Orienté Objet

Approches orientées objet (Rappel)

Approches classiques

Séparation nette entre les données et les procédures de traitement de ces données



Au moindre changement des données, il faut modifier les procédures de manipulation de ces données

Approches orientées objet (Rappel)

Approches classiques



Difficulté de maintenance !!



Solution : Approche Orienté Objet

Approches orientées objet

(Rappel)

- Programme : ensemble d'objets indépendants communiquant par envoi de messages
- Lorsqu'un objet désire une information d'un autre objet, il lui envoi un message
- Les messages sont le seul moyen de communication entre objets
- Un objet regroupe ses données et les procédures de manipulation de ces données

Approches orientées objet (Rappel)

- Programme = ensemble d'objets
- Objet = données + procédure
- Communication = envoi de message

Approches orientées objet (Rappel)

L'orienté objet repose sur trois concepts de base :

- ❑ Le concept de structuration en utilisant les **objets** et les **classes**
- ❑ Le concept de communication par envoi de **messages**
- ❑ Le concept de construction par affinage avec l'**héritage**.

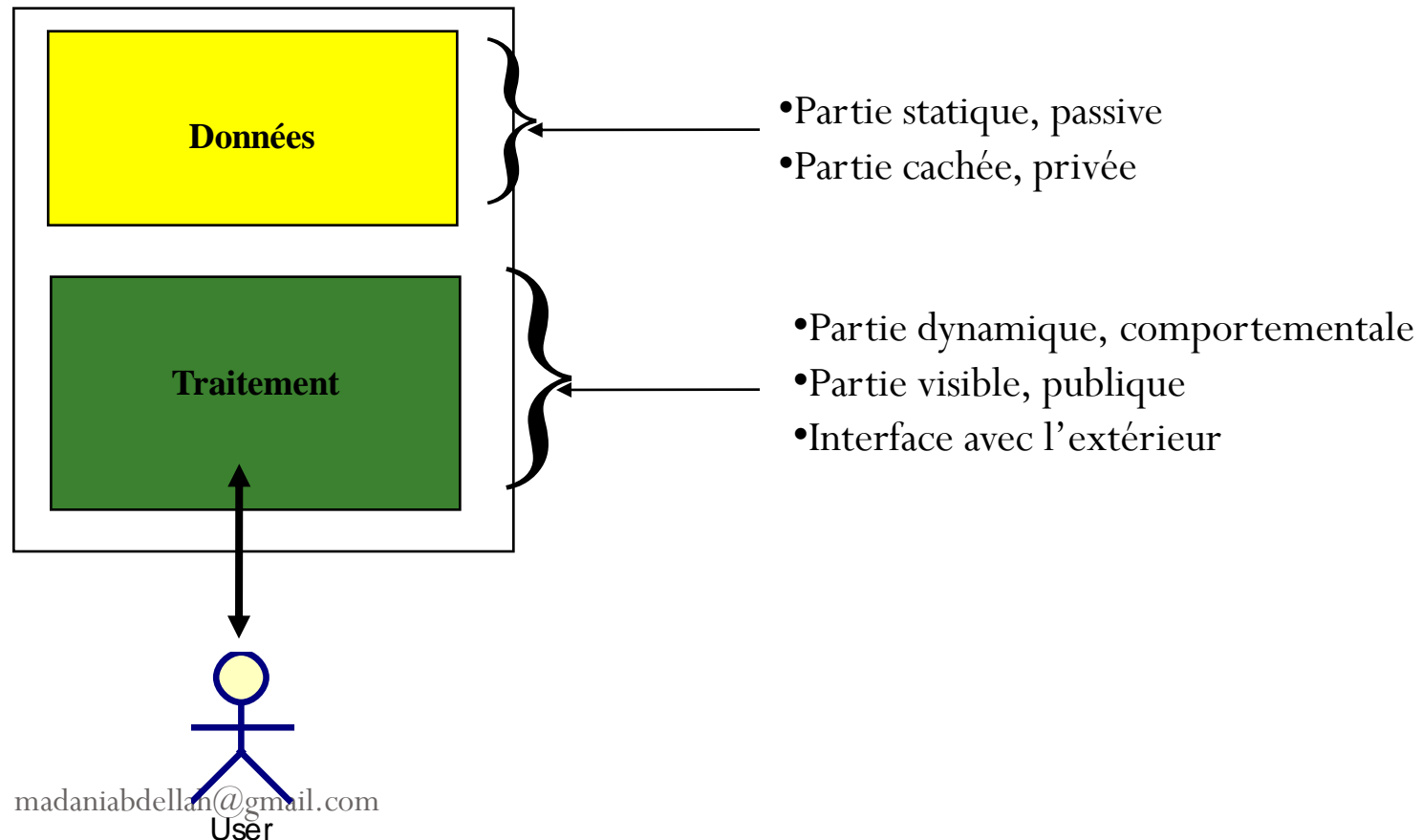
Approches orientées objet (Rappel)

- Un objet est une entité
 - ❑ Autonome (ayant des limites bien définies)
 - ❑ Ayant une existence propre
 - ❑ Ayant un sens dans le domaine étudié
- Un objet regroupe à la fois
 - ❑ Des données (attributs)
 - ❑ Des procédures (opérations)

Notion d'Encapsulation

Approches orientées objet

(Rappel)



Approches orientées objet

(Rappel)

- Les données de l'objet constituent sa partie privée, c'est-à-dire la partie cachée
- Les opérations sont la partie accessible depuis l'extérieur : pour accéder aux données de l'objet on est obligé de passer par les méthodes

Interface de l'objet

Approches orientées objet

(Rappel)

- Plusieurs objets peuvent avoir des données et des comportements (opérations) en commun

Il devient nécessaire de les regrouper dans un modèle général :
classe d'objets

- Une classe est un ensemble d'objets ayant les mêmes données et les mêmes opérations

Approches orientées objet

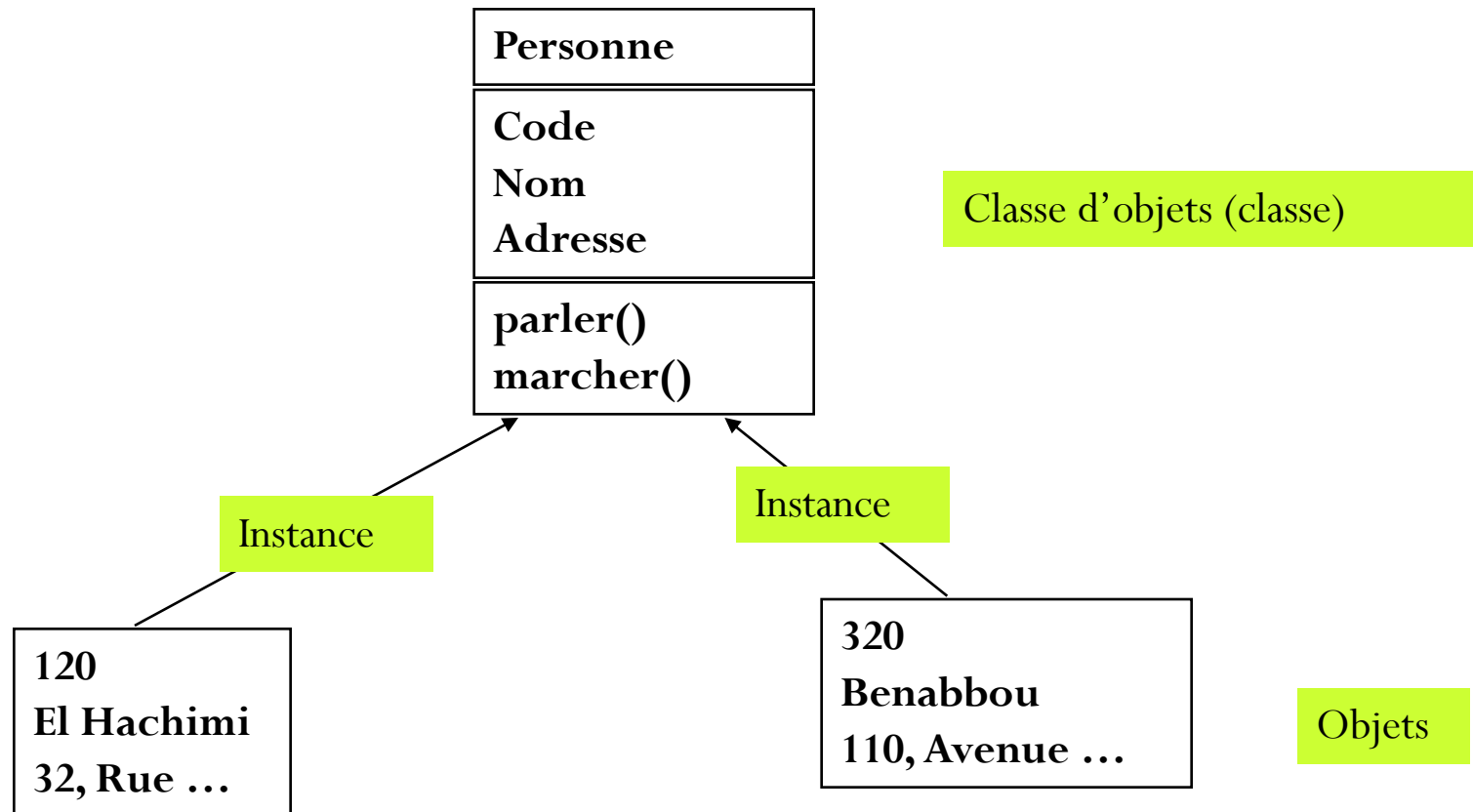
(Rappel)

- Une classe est un moule à partir de laquelle on crée (instancie) des objet



- les objets sont des instances d'une classe
- L'instanciation est la création d'un objet d'une classe

Approches orientées objet (Rappel)



Approches orientées objet

(Rappel)

Remarques

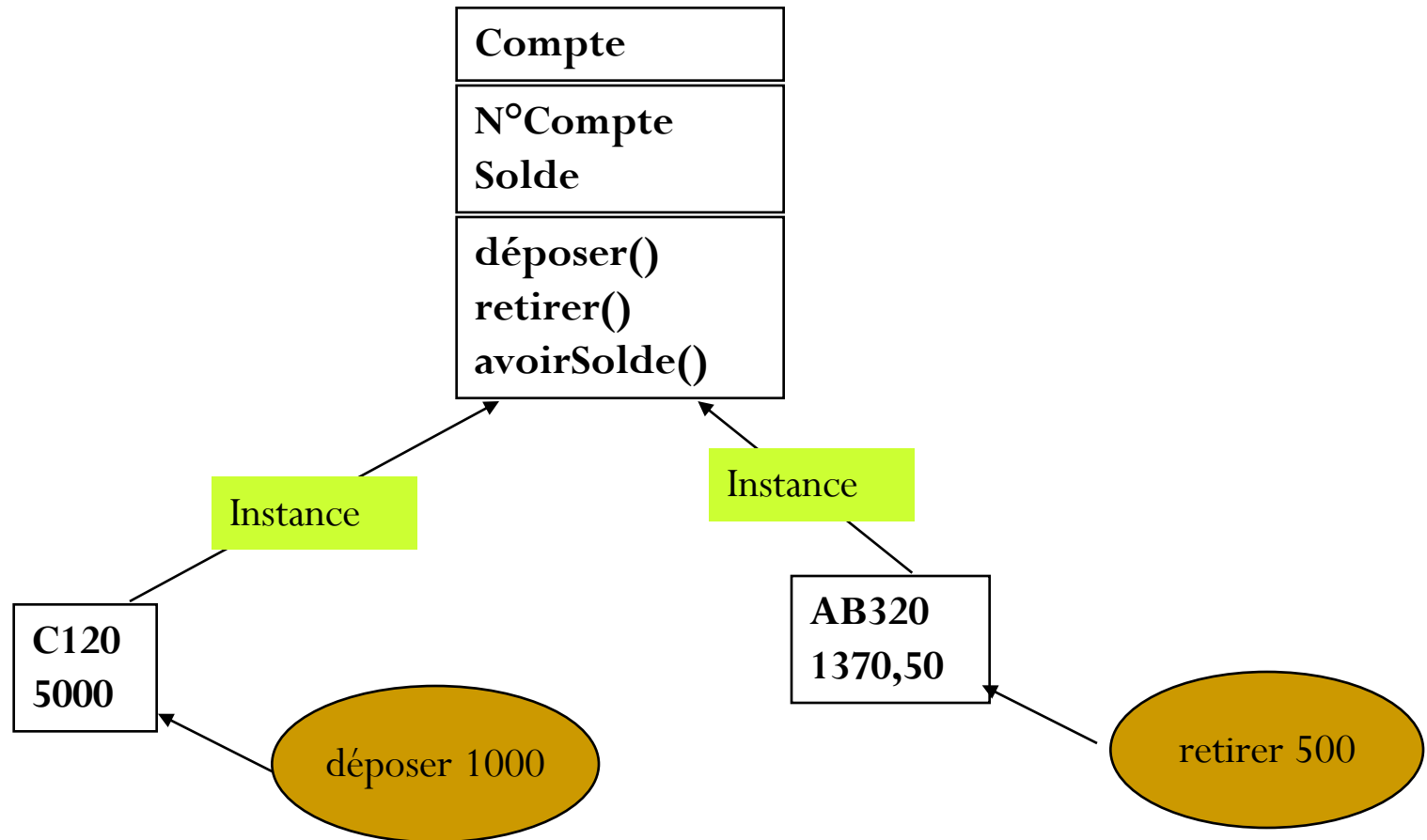
- Classe : modèle général
- Objets :
 - ❑ cas particuliers
 - ❑ Instances
 - ❑ occurrences

Approches orientées objet

(Rappel)

- Les objets communiquent par envoi de messages
- Envoyer un message à un objet, c'est lui demander un service
- Lorsqu'un objet reçoit un message :
 - Soit le message correspond à un traitement défini dans la classe de l'objet auquel cas la méthode correspondante est exécutée.
 - Soit le message ne correspond pas, l'objet refuse le message et signale une erreur

Approches orientées objet (Rappel)



Généralisation / Spécialisation et héritage

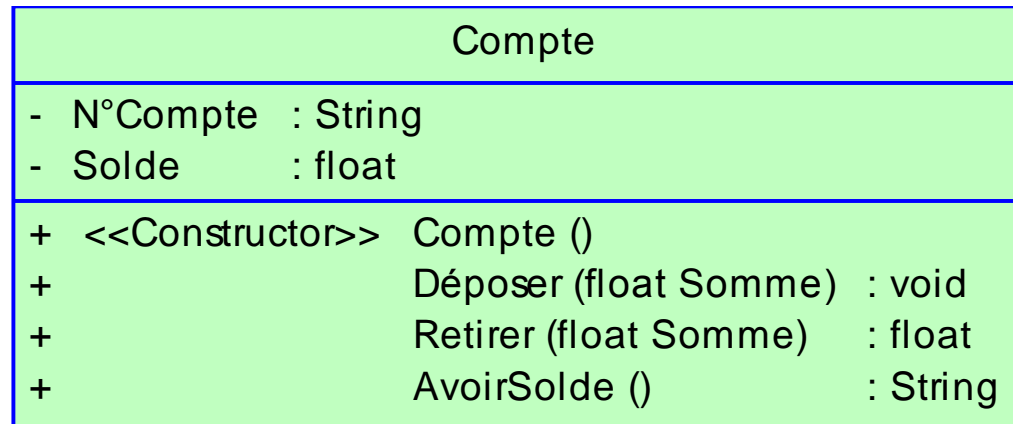
- La généralisation est la relation entre une classe et une ou plusieurs de ses versions raffinées.
- On appelle la classe dont on tire les précisions la super-classe et les autres classes les sous-classes.
- C'est une relation de type « est un (is a) » ou « est une sorte de ».

Généralisation / Spécialisation et héritage

La classe spécialisée (sous-classe)

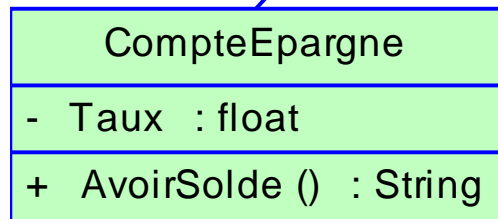
- hérite les méthodes et les attributs de la classe générale (super-classe)
- peut ajouter ses propres attributs et méthodes.
- peut redéfinir le comportement d'une méthode, mais pas des attributs.

Généralisation / Spécialisation et héritage



Dans la classe CompteEpargne :

- N°Compte et Solde : hérités
- Déposer(), Retirer() : héritées
- avoirSolde() est une méthode redéfinie



Interprétation des messages

Lorsqu'un objet reçoit un message :

- Si sa classe contient une méthode de même nom, elle sera exécutée
- Sinon, on remonte dans la hiérarchie à la recherche d'une méthode de même nom.

Interprétation des messages

Exemple

Soit CE une instance de la classe CompteEpargne :

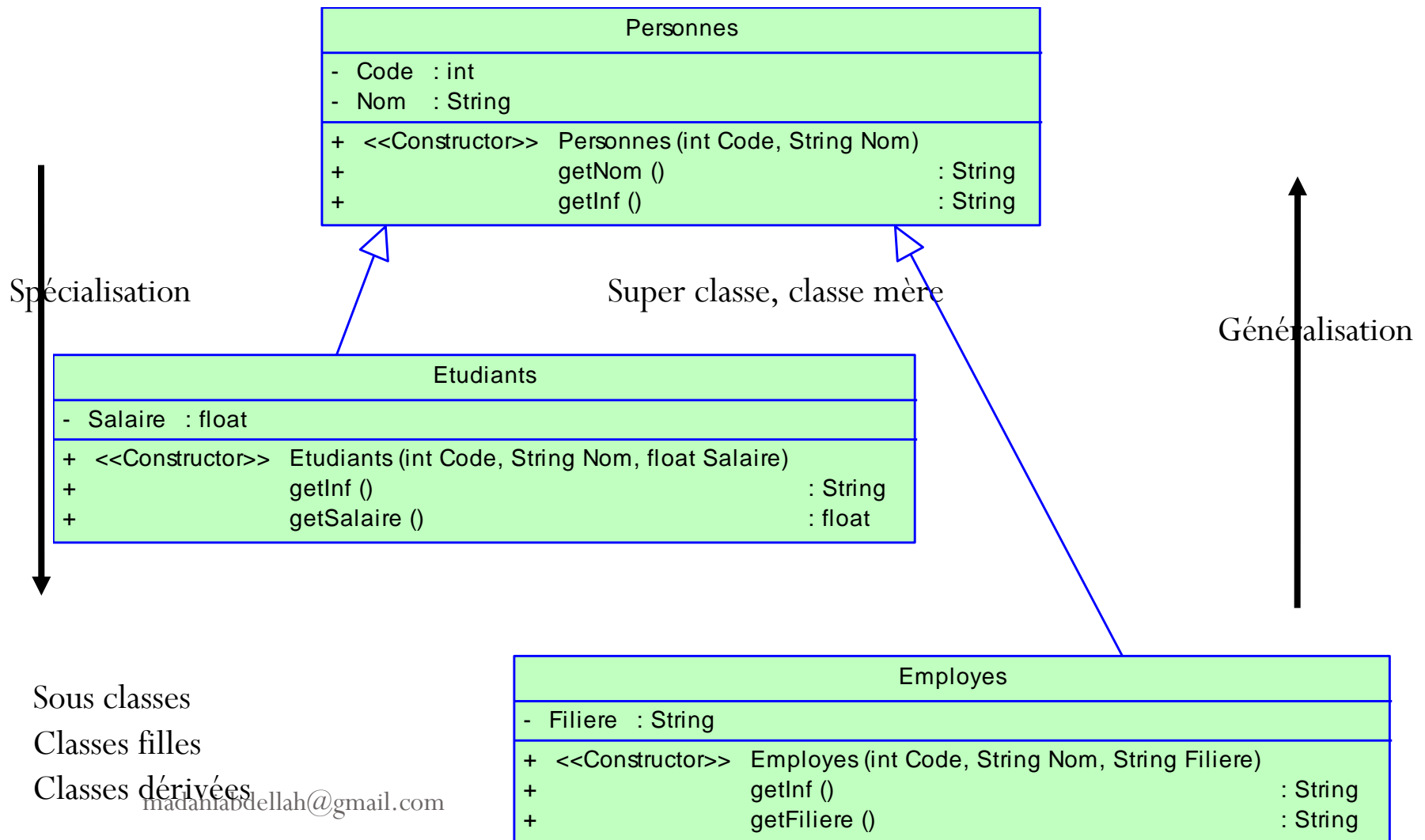
- ❑ C.déposer() : fait appel à la méthode 'déposer' de la classe 'Compte'
- ❑ C.avoirSolde() : fait appel à la méthode 'avoirSolde' de la classe 'CompteEpargne'

Généralisation / Spécialisation et héritage

Remarques

- La généralisation et la spécialisation sont deux façons pour voir la même relation, top-down (spécialisation) ou bottom-up (généralisation).
- L'héritage est l'implémentation de la relation de la généralisation/spécialisation.
- Une classe Java ne peut hériter que d'une seule classe.

Généralisation / Spécialisation et héritage



Création d'une classe

```
class Compte {  
    // déclaration des données (privées)  
    private String NCompte;  
    private float Solde;  
    ...  
}
```


Création d'une classe

```
class Compte{  
    ...  
    //Déclaration des méthodes (publics)  
    public Compte(String n, float s){NCompte=n;Solde=s;}  
    public void deposer(float somme){Solde+=somme;}  
    public void retirer(float somme){  
        if (Solde<somme)  
            Console.Out.WriteLine(«Solde insuffisant ! »);  
        else  
            Solde-=somme;  
    }  
    public float avoirSolde(){return Solde;}  
}
```


Création d'une classe

- Les données sont généralement privées.
- Les méthodes sont généralement publiques (interface)
- Le constructeur est une méthode spéciale :
 - ❑ Porte le même nom que la classe
 - ❑ Utilisée pour initialiser les objets
 - ❑ Ne retourne aucune valeur, même void
 - ❑ Invoquée implicitement au moment de la création des objets
 - ❑ On peut avoir plusieurs constructeurs

Création d'objets (instances)

```
class test{  
    public static void Main(String arg[]) {  
        Compte c1=new Compte("C120",5000);  
        Compte c2;  
        c2=new Compte("AB320",1370.50);  
        float s=c1.Solde;       Erreur, car ...  
        float s1=c1.avoirSolde();  
        Console.Out.WriteLine("Solde "+s1);  
        c2.deposer(500);  
        Console.Out.WriteLine("Solde "+c2.avoirSolde());  
        ...  
    }  
}
```

Création d'objets (instances)

- Le mot clé **this** désigne l'objet courant (l'objet qui fait appel à la méthode)

- Par exemple

```
public void deposer(float somme) {  
    /*augmenter le solde de l'objet faisant appel à la méthode déposer()*/  
    This.Solde+=somme;  
}  
...  
Compte c1=new Compte(100,3500)  
c1.deposer(2000); // ➡ this indique l'objet c1
```

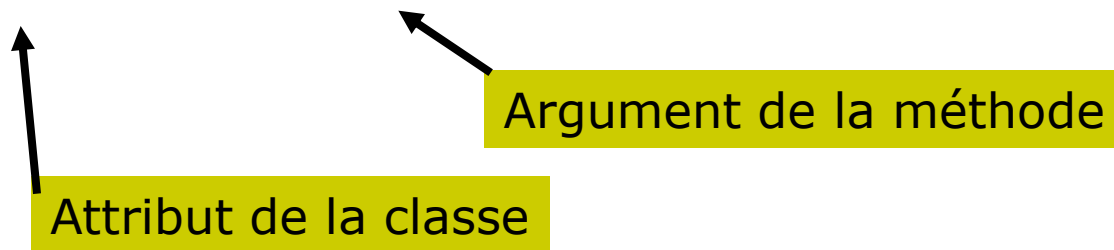
Création d'objets (instances)

- Remarque

Le mot clé `this` est obligatoire lorsqu'il y a conflit entre un attribut de classe et un argument de la méthode utilisant cet attribut

- Par exemple :

```
public Compte(String NCompte, float Solde) {  
    this.NCompte=NCompte;  
    this.Solde=Solde;  
}
```



Données de classes/d'instances

Une donnée (attribut ou méthode) peut être :

- Donnée d'instance, propre à l'objet
- Donnée de classe, partagée entre tous les objets de la classe

Données de classes/d'instances

Une donnée d'instances

- Attribut : chaque objet possède sa propre valeur
- Méthode : invoquée en faisant référence à l'objet

Données de classes/d'instances

Une donnée de classe (déclaration précédée du mot clé "static")

- Attribut : partagée entre tous les objets de la classe.
- Méthode : invoquée en précisant la classe à la place de l'objet

Données statiques

```
class Personne {  
    //attributs d'instances  
    private int Code;  
    private String Nom;  
    //attribut de classe (statique)  
    private static int NbreEmp;  
    ...  
}
```


Données statiques

```
class Personne {  
    ...  
    public Personne(int code, String nom) {  
        NbrePers++; Code=code; Nom=nom;  
    }  
    public String getInf() {  
        return "Code "+Code+"\nNom "+Nom;  
    }  
    public static int getNbre() {  
        return NbrePers;  
    }  
}
```

Données statiques

```
class test {  
    public static void Main(String arg[]) {  
        Personne p=new Personne(10, "A. B");  
        Personne q=new Personne(20, "I. I");  
        Console.Out.WriteLine(p.getInf());  
        Console.Out.WriteLine(Personne.getNbre());  
        Console.Out.WriteLine(q.getInf());  
        Console.Out.WriteLine(Personne.getNbre());  
    }  
}
```

Notion d'héritage

L'héritage est le mécanisme de partage des informations entre objets tout en préservant leurs différences

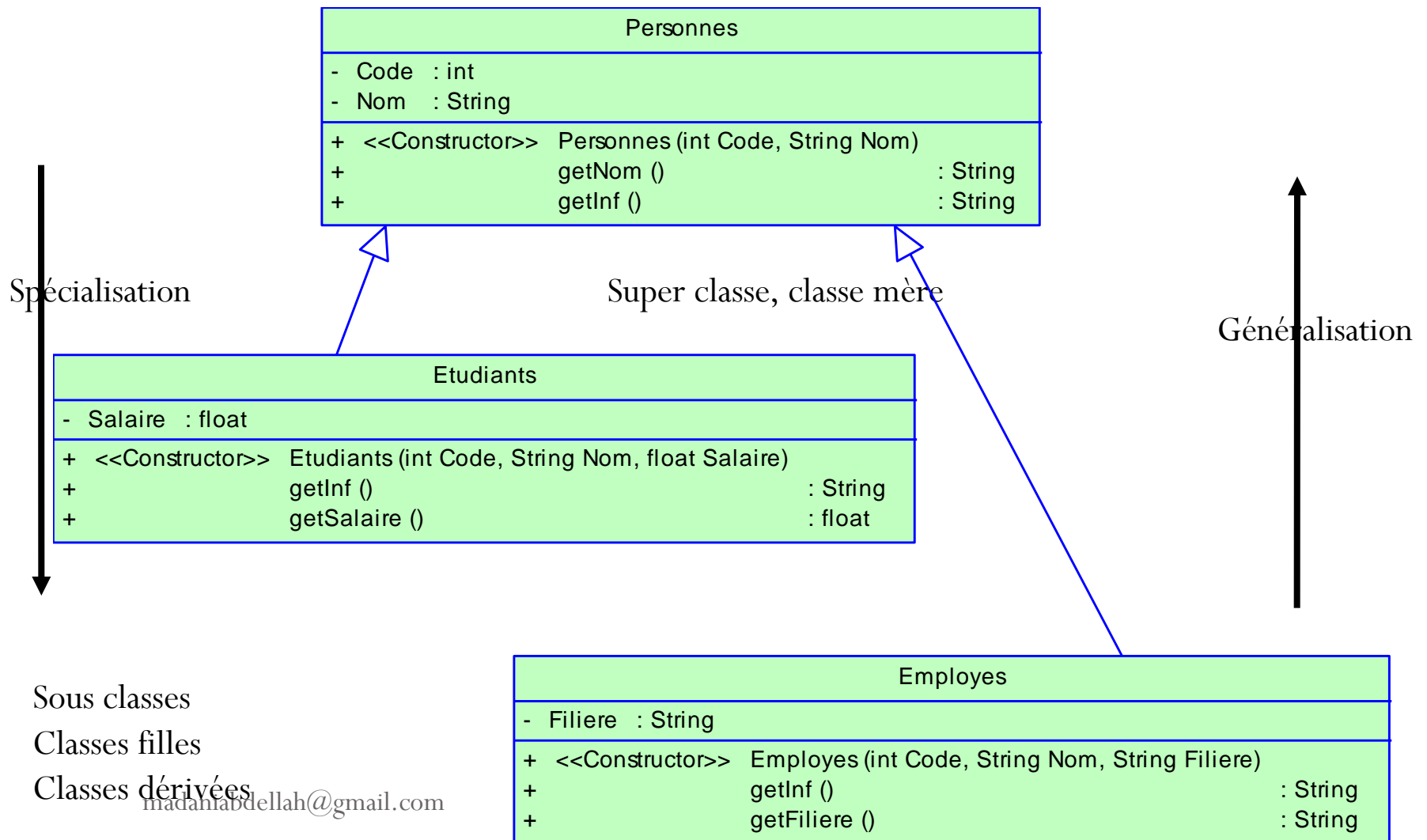
Notion d'héritage

Personnes		
-	Code	: int
-	Nom	: String
+	<<Constructor>> Personnes (int Code, String Nom)	
+	getNom ()	: String
+	getInf ()	: String

Etudiants		
-	Code	: int
-	Nom	: String
-	Salaire	: float
+	<<Constructor>> Etudiants (int Code, String Nom, float Salaire)	
+	getNom ()	: String
+	getInf ()	: String
+	getSalaire ()	: float

Employes		
-	Code	: int
-	Nom	: String
-	Filiere	: String
+	<<Constructor>> Employes (int Code, String Nom, String Filiere)	
+	getNom ()	: String
+	getInf ()	: String
+	getFiliere ()	: String

Notion d'héritage



Notion d'héritage

```
class Personne {  
    private int Code;  
    private String Nom;  
    public Personne(int code, String nom) {  
        Code=code; Nom=nom;  
    }  
    public String getNom() {  
        return Nom;  
    }  
    public String getInf() {  
        return "Code : "+Code + "\nNom : Nom";  
    }  
}
```

Notion d'héritage

```
class Employe : Personne {  
    private float Salaire;  
    public Employe(int code, String nom, float salaire):base(code,nom) {  
        Salaire=salaire;  
    }  
    public float getSalaire() {  
        return Salaire;  
    }  
    public String getInf() {String inf=base.getInf();  
        Inf=inf + "\nSalaire :"+Salaire;  
        return Inf;  
    }  
}
```

Notion d'héritage

```
class Etudiant : Personne {  
    private float Salaire;  
    public Etudiant(int code, String nom, String filiere):base(code,nom) {  
        Filiere=filiere;  
    }  
    public float getFiliere() {  
        return Filiere;  
    }  
    public String getInf() {String inf=base.getInf();  
        Inf=inf + "\nFiliere :"+Filiere;  
        return Inf;  
    }  
}
```


Notion d'héritage

Quatre types de visibilité de données

- ❑ Privée : donnée visible que par les méthodes de la classe
- ❑ Protégée : donnée visible que par les méthodes de la classes et des sous classes directes
- ❑ Publique : donnée visible depuis n'importe quelle méthode