

Programmation orientée objet

Présentation : Tom Niget

Mail de contact : polytech@niget.fr

Site et ressources : niget.fr

Cours réalisé grâce notamment aux ressources de :
Julien Deantoni, Benjamin Werner, Joseph Albahari

Programmation orientée objet

- Paradigme de programmation : façon de programmer
 - **Programmation impérative** : on donne des ordres à la machine, la machine obéit
 - Assembleur, BASIC, C, Python, Java, C++, C#, JavaScript, ...
 - **Programmation déclarative** : on décrit un résultat à la machine, la machine se débrouille pour nous le donner
 - SQL, CSS, HTML, Prolog, ...
- (liste non exhaustive)

Programmation orientée objet

- **Programmation impérative** : on donne des ordres à la machine, la machine obéit
 - Assembleur, BASIC (Casio, TI, ...), ...
 - **Programmation procédurale** : on divise le programme en sous-programmes (procédures, aussi appelées fonctions) pour pouvoir réutiliser des morceaux de logique
 - C, Python, Java, C++, C#, JavaScript, ...

Programmation orientée objet

- **Programmation procédurale** : on divise le programme en sous-programmes (procédures, aussi appelées fonctions) pour pouvoir réutiliser des morceaux de logique
 - C, Pascal, Fortran, ...
 - **Programmation orientée objet** : on encapsule les données et les comportements des entités
 - Java, C++, C#, ...

Programmation orientée objet

```
float solde = 100;  
solde = solde + 50;  
solde = solde - 20;  
solde = solde - 10;  
solde = solde + 8;  
solde = solde - 1.99;  
solde = solde * 1.01;
```

Impératif

- Code très concis
- Duplication de code
- Que « fait » chaque ligne ?

Encapsulation des comportements

```
float credit(float solde, float montant) {  
    return solde + montant;  
}  
  
float debit(float solde, float montant) {  
    return solde - montant;  
}  
  
float frais(float solde, float frais) {  
    return debit(solde, frais);  
}  
  
float interets(float solde, float taux) {  
    return credit(solde, solde * taux);  
}
```

Procédural

- Chaque comportement est isolé
- On voit quelle opération est faite grâce au nom de la fonction
- Rien n'empêche de modifier le solde à la main

```
float solde = 100;  
float frais_bancaires = 1.99;  
float taux_interets = 0.01;  
solde = credit(solde, 50);  
solde = debit(solde, 20);  
solde = debit(solde, 10);  
solde = credit(solde, 8);  
solde = frais(solde, frais_bancaires);  
solde = interets(solde, taux_interets);
```

Programmation orientée objet

```
float credit(float solde, float montant) {  
    return solde + montant;  
}
```

```
float debit(float solde, float montant) {  
    return solde - montant;  
}
```

```
float frais(float solde, float frais) {  
    return debit(solde, frais);  
}
```

```
float interets(float solde, float taux) {  
    return credit(solde, solde * taux);  
}
```

```
float solde = 100;  
float frais_bancaires = 1.99;  
float taux_interets = 0.01;  
solde = credit(solde, 50);  
solde = debit(solde, 20);  
solde = debit(solde, 10);  
solde = credit(solde, 8);  
solde = frais(solde, frais_bancaires);  
solde = interets(solde, taux_interets);
```

```
float debit(float solde, float montant) {  
    if (montant > solde) {  
        throw new Exception("Solde insuffisant");  
    }  
    return solde - montant;  
}
```

```
float solde = 100;  
solde = debit(solde, 9000); // Exception: Solde insuffisant  
solde = solde - 9000; // Pas d'exception, solde négatif
```

Programmation orientée objet

Données
(variables)

```
class Compte {  
    private float solde;
```

Comportements
(fonctions / méthodes)

```
    public Compte(float soldeInitial) { this.solde = soldeInitial; }  
  
    public void Credit(float montant) { this.solde += montant; }  
    public void Debit(float montant) {  
        if (montant > this.solde) {  
            throw new Exception("Solde insuffisant");  
        }  
        this.solde -= montant;  
    }  
  
    public void Frais(float frais) { this.Debit(frais); }  
    public void Interets(float taux) { this.Credit(this.solde * taux); }  
}
```

`private` = inaccessible depuis
l'extérieur

`this` sert à accéder aux membres de la classe

Programmation orientée objet

```
class Compte {  
    private float solde;  
  
    public Compte(float soldeInitial) { this.solde = soldeInitial; }  
  
    public void Credit(float montant) { this.solde += montant; }  
    public void Debit(float montant) {  
        if (montant > this.solde) {  
            throw new Exception("Solde insuffisant");  
        }  
        this.solde -= montant;  
    }  
  
    public void Frais(float frais) { this.Debit(frais); }  
    public void Interets(float taux) { this.Credit(this.solde * taux); }  
}
```

```
Compte compte = new Compte(100);  
compte.Debit(9000); // Exception: Solde insuffisant  
compte.solde = 1234; // Erreur de compilation: solde est privé
```

On n'aura jamais de solde négatif !
À moins que...

Programmation orientée objet

```
class Compte {  
    private float solde;  
  
    public Compte(float soldeInitial) { this.solde = soldeInitial; }  
  
    public void Credit(float montant) { this.solde += montant; }  
    public void Debit(float montant) {  
        if (montant > this.solde) {  
            throw new Exception("Solde insuffisant");  
        }  
        this.solde -= montant;  
    }  
  
    public void Frais(float frais) { this.Debit(frais); }  
    public void Interets(float taux) { this.Credit(this.solde * taux); }  
}
```

```
Compte compte = new Compte(100);  
compte.Credit(-9000); // Pas d'exception, solde négatif  
compte.Interets(-1000); // Pas d'exception, solde négatif
```

Programmation orientée objet

```
class Compte {  
    private float solde;  
  
    public Compte(float soldeInitial) { this.solde = soldeInitial; }  
  
    public void Credit(float montant) { this.solde += montant; }  
    public void Debit(float montant) {  
        if (montant > this.solde) {  
            throw new Exception("Solde insuffisant");  
        }  
        this.solde -= montant;  
    }  
  
    public void Frais(float frais) { this.Debit(frais); }  
    public void Interets(float taux) { this.Credit(this.solde * taux); }  
}
```

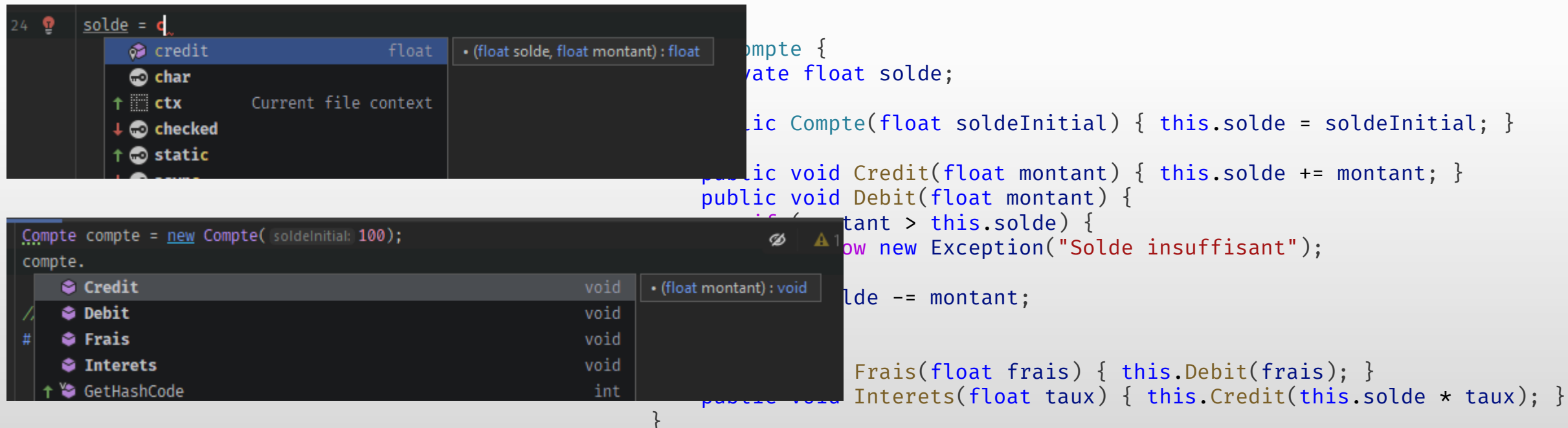
↑ Exemple simplifié. Dans un vrai programme comptable,

- Le solde commence à 0. L'argent n'apparaît pas de nulle part.
- Une opération se fait entre deux comptes : virement, prélèvement, paiement.

Programmation orientée objet

Avantages de la version objet :

- **Encapsulation des données** : on ne peut pas mettre n'importe quoi dans le solde.
- **Comportements bien séparés** dans le code : facile de repérer quel code fait quoi.
- **Découvrabilité** : l'IDE nous indique les opérations disponibles.



The screenshot shows an IDE with two code completion windows. The top window is triggered by the text 'solde =' and lists options: 'credit' (float), 'char', 'ctx' (Current file context), 'checked', and 'static'. The bottom window is triggered by 'compte =' and lists options: 'Credit' (void), 'Debit' (void), 'Frais' (void), 'Interets' (void), and 'GetHashCode' (int). The background code is a Java-like snippet for a 'Compte' class.

```

24  solde = d
    credit float • (float solde, float montant) : float
    char
    ctx Current file context
    checked
    static

Compte compte = new Compte( soldeInitial: 100);
compte.
    Credit void • (float montant) : void
    Debit void
    Frais void
    Interets void
    GetHashCode int

Compte {
    private float solde;

    public Compte(float soldeInitial) { this.solde = soldeInitial; }

    public void Credit(float montant) { this.solde += montant; }
    public void Debit(float montant) {
        if (montant > this.solde) {
            throw new Exception("Solde insuffisant");
        }
        this.solde -= montant;
    }

    void Frais(float frais) { this.Debit(frais); }
    void Interets(float taux) { this.Credit(this.solde * taux); }
}

```

Programmation orientée objet

```
class Compte {
    private float solde;

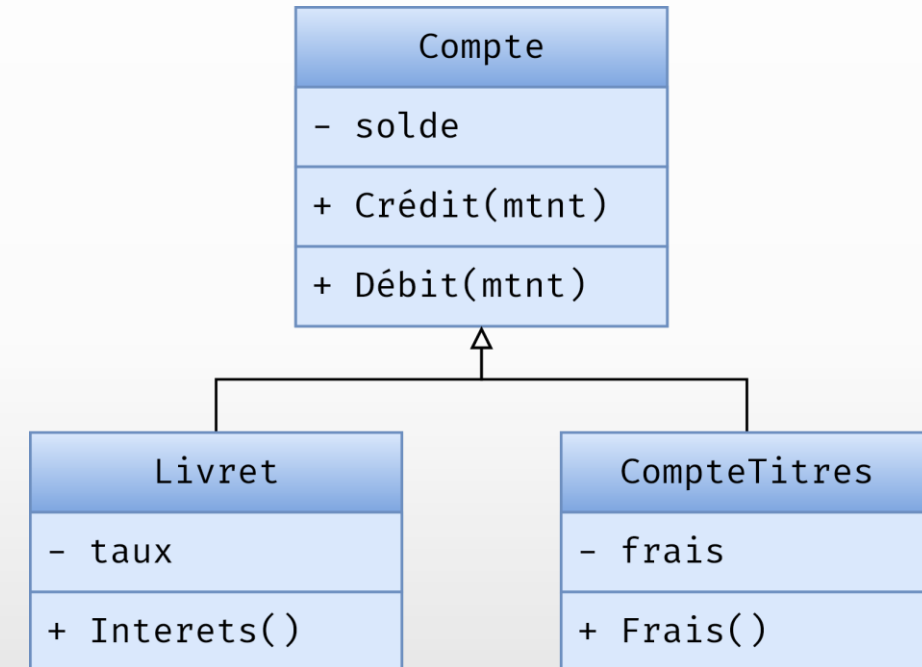
    public Compte(float soldeInitial) { this.solde = soldeInitial; }

    public void Credit(float montant) { this.solde += montant; }
    public void Debit(float montant) {
        if (montant > this.solde) {
            throw new Exception("Solde insuffisant");
        }
        this.solde -= montant;
    }

    public void Frais(float frais) { this.Debit(frais); }
    public void Interets(float taux) { this.Credit(this.solde * taux); }
}
```

Différents types de comptes (exemple simplifié) :

- Compte courant basique
- Livret : avec intérêts annuels
- Compte titres : avec frais de tenue de compte



+ public
- private

Programmation orientée objet

Privé mais
accessible par
les classes filles

```
class Compte {
    protected float solde;
    ...
}
```

```
class Livret : Compte {
    private float taux;

    public Livret(float soldeInitial, float taux) : base(soldeInitial) {
        this.taux = taux;
    }

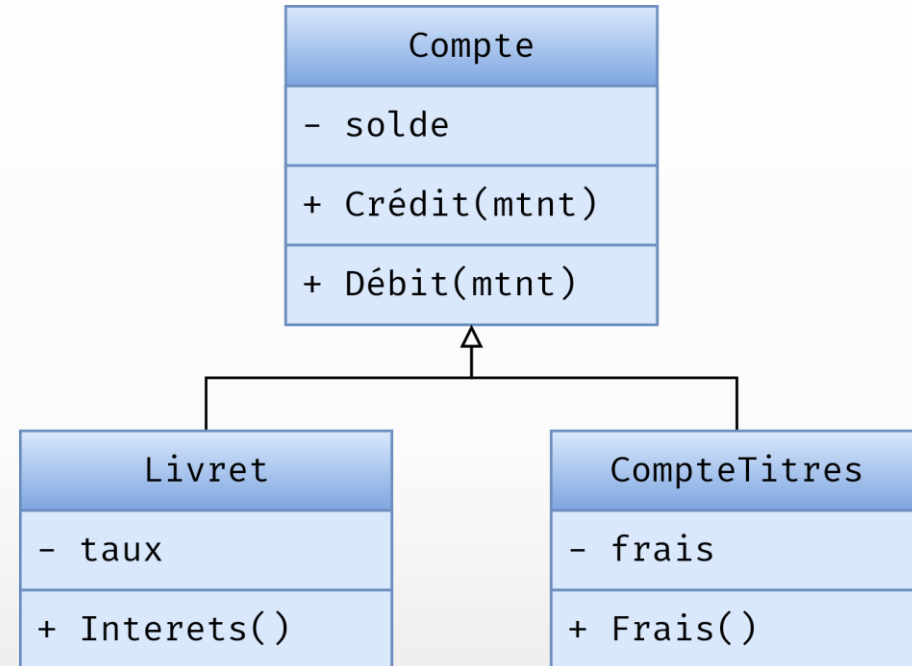
    public void Interets() { this.Credit(this.solde * this.taux); }
}
```

```
class CompteTitres : Compte {
    private float frais;

    public CompteTitres(float soldeInitial, float frais) : base(soldeInitial) {
        this.frais = frais;
    }

    public void Frais() { this.Debit(this.frais); }
}
```

Héritage de classes



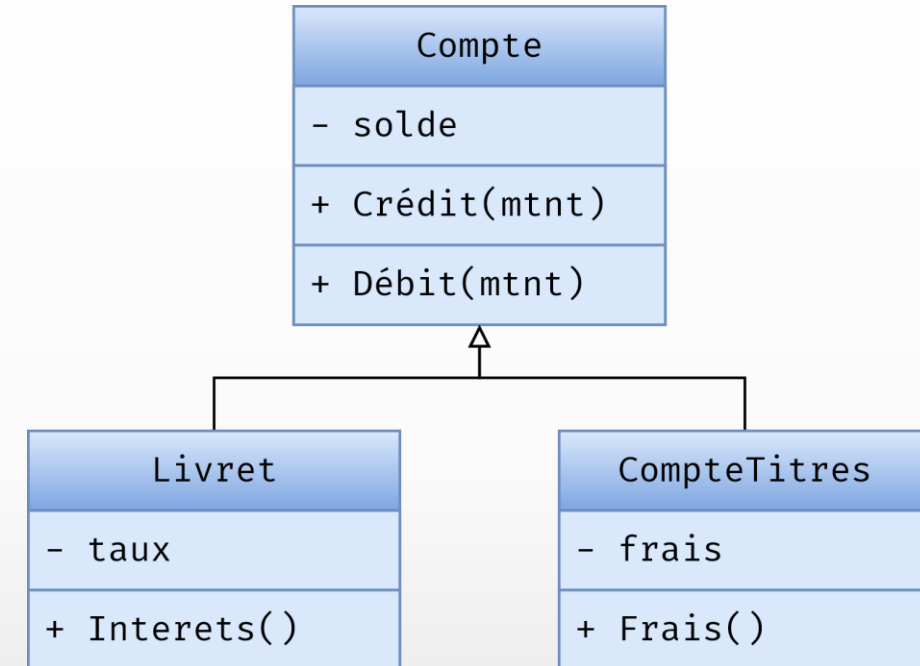
Principe de substitution de Liskov :

Si la classe B hérite de la classe A (= est une sous-classe de A), alors partout où on attend des objets A, on peut passer des objets B.

Programmation orientée objet

```
void TraitementQuelconque(Compte cpt)
{
    // traitement indifférent au type de compte
    cpt.Credit(100);
}
```

```
Compte compte1 = new Compte(100);
Compte compte2 = new Livret(100, 0.01);
Compte compte3 = new CompteTitres(100, 1.99);
TraitementQuelconque(compte1);
TraitementQuelconque(compte2);
TraitementQuelconque(compte3);
```



Héritage : relation « **est un** ». Un livret est un compte (= est un type particulier de compte).
Mais tout compte n'est pas forcément un livret !

Principe de substitution de Liskov :
Si la classe B hérite de la classe A (= est une sous-classe de A), alors partout où on attend des objets A, on peut passer des objets B.

Programmation orientée objet

```
void TraitementQuelconque(Compte cpt)
{
    // traitement indifférent au type de compte
    cpt.Credit(100);
}
```

```
Compte compte1 = new Compte(100);
Compte compte2 = new Livret(100, 0.01);
Compte compte3 = new CompteTitres(100, 1.99);
TraitementQuelconque(compte1);
TraitementQuelconque(compte2);
TraitementQuelconque(compte3);
```

Polymorphisme : le fait de pouvoir traiter différents types de choses avec une **interface commune**

Interface : surface publique permettant d'interagir avec quelque chose = méthodes publiques, etc.

Une interface est un **contrat**, une **promesse**, entre plusieurs parties du programme. L'interface de Compte contient la méthode Credit = promesse que tout compte (y compris Livret, ...) peut recevoir un Credit.

Programmation orientée objet

- On décrit les entités comme des **classes**
- Les classes **encapsulent** les **données** et les **comportements**
- Le **polymorphisme** permet de mettre en commun des classes liées entre elles
- L'approche augmente la **lisibilité** et la **maintenabilité** du code

Le langage C#

- Appartient à la « **famille C** » comme le Java (très similaire), le C++, le JavaScript, le PHP
 - Code groupé avec des accolades, instructions terminées par des points-virgules
- Créé au début des années 2000 pour concurrencer le Java
- **Statiquement typé** : les éléments (méthodes, variables, paramètres) ont un type fixe défini lors de la compilation
- **Managé** : les programmes C# s'exécutent dans un environnement d'exécution (.NET)
 - Pas de pointeurs, pas d'accès direct à la mémoire, pas de segfaults...
 - Portable : un même programme marchera sous Windows, Linux, macOS

Le langage C#

- **Langage entièrement objet** : tous les types héritent de la classe **Object**
 - En Java, les types primitifs (entiers, ...) ne sont pas des objets, et on a que `int != Integer, boolean != Boolean, ...`
- **Gestion automatique de la mémoire** : pas de `malloc/free`, les objets sont alloués lors de leur création et libérés quand ils ne sont plus utilisés
- **Outils open-source et multi-plateformes** : il existe plusieurs compilateurs, IDEs, analyseurs, ...

Le langage C#

- Types de base similaires au C : entiers (byte, short, int, long) ; réels (float, double, decimal), bool, char, string (!)
 - Contrairement au C, les chaînes de caractères sont supportées d'emblée avec le type string. Une string est une séquence de char (caractères).
- Types définis par l'utilisateur : les classes (on verra le reste plus tard)
- Une classe doit contenir un constructeur : fonction qui sert à créer des nouvelles instances de la classe (= des objets).
 - On appelle le constructeur via l'opérateur new

```
public Compte(float soldeInitial) {  
    this.solde = soldeInitial;  
}  
...  
new Compte(123)
```

Le langage C#

TD 1