

# Module 7:

## Programmation orientée objet

### Vue d'ensemble

- Les classes et les objets
- Utilisation de l'encapsulation
- C# et l'orientation objet
- Définition des systèmes orientés objet

## Les classes et les objets

- Qu'est-ce qu'une classe?
- Qu'est-ce qu'un objet?
- Comparaison Classes de Structures
- Abstraction

## Qu'est-ce qu'une classe?

- Une classe est un modèle de construction d'objets
- C'est dans la classe que sera défini les particularités de chaque objet
- Il faut définir:
  - Les données de chaque instance de cette classe (les propriétés)
  - Les actions/tâches pouvant être faite sur chaque instance (les méthodes)
  - les comportements internes à l'objet

## Qu'est-ce qu'un objet?

- Un objet est une instance d'une classe
- Les objets exposent:
  - Identité: Objets se distinguent les uns des autres
  - Comportement: les actions/tâches définies dans la classe
  - L'état: les informations de l'objet



## Comparaison Classes de Structures

- Une structure est un modèle pour une valeur
  - Pas d'identité, données accessible, aucun comportement supplémentaire
- Une classe est un modèle pour un objet
  - Identité, données inaccessible, comportement supplémentaire possible

```
struct Time                class BankAccount
{
    public int hour;        {
    public int minute;      ...
}                          ...
}
```

## Abstraction

- Abstraction est l'ignorance sélective
  - Décider ce qui est important et ce qui n'est pas
  - Cibler et dépendre de ce qui est important
  - Ignorer et ne pas dépendre de ce qui est sans importance
  - Utilisez l'encapsulation afin d'appliquer une abstraction

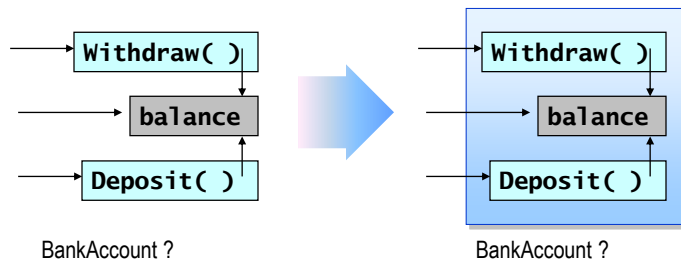
**The purpose of abstraction is not to be vague,  
but to create a new semantic level in which one can be absolutely precise.  
Edsger Dijkstra**

## Utilisation de l'encapsulation

- La combinaison de données et méthodes
- Contrôle de la visibilité d'accès
- Pourquoi encapsuler?
- Données des objets
- Utilisation des données statiques
- Utilisation des méthodes statiques

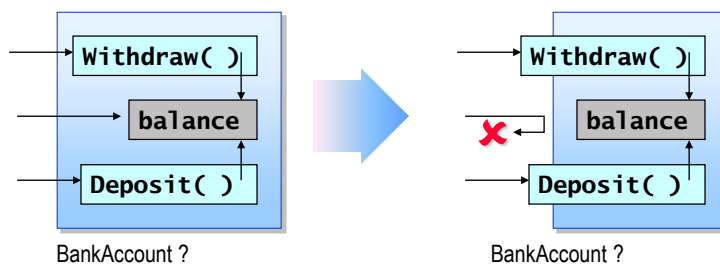
## La combinaison de données et méthodes

- Combiner les données et les méthodes en une seule *entité*



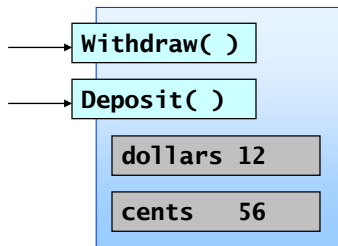
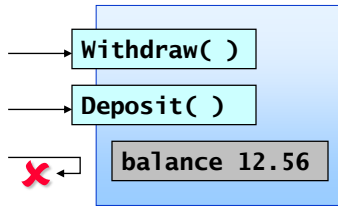
## Contrôle de la visibilité d'accès

- Les méthodes sont *public*, Accessible de l'extérieur
- Les données sont *privé*, Accessible uniquement de l'intérieur



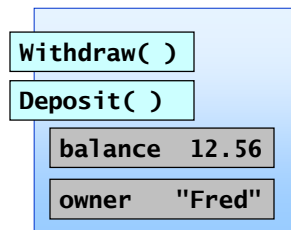
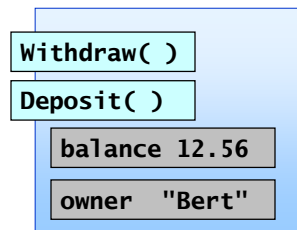
## Pourquoi Encapsuler?

- Permet le contrôle
  - L'utilisation de l'objet est uniquement faite à travers les méthodes publiques
- Permet le changement
  - L'utilisation de l'objet n'est pas affectée si la structure des données privées changent



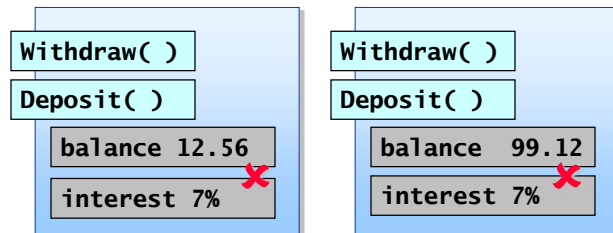
## Données des objets

- Les données de l'objet décrivent les informations pour chaque objet *individuel*
  - Par exemple, chaque compte bancaire a son propre solde. Si deux comptes ont le même solde, ce n'est qu'une coïncidence...



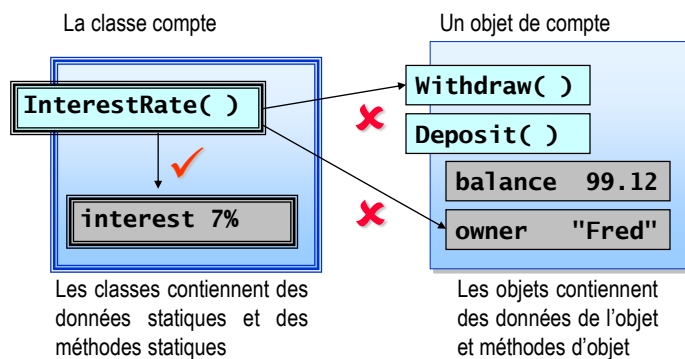
## Utilisation des données statiques

- Les données statiques décrivent des informations pour tous les objets d'une classe
  - Par exemple, supposons que tous les comptes partagent le même taux d'intérêt. Stocker le taux d'intérêt dans chaque compte serait une mauvaise idée. Pourquoi?



## En utilisant des méthodes statiques

- Les méthodes statiques ne peuvent accéder qu'à des données statiques
  - Une méthode statique est appelée sur la classe, pas sur un objet



## C # et l'orientation objet

- Hello, World Revisited
- Définition de classes simples
- L'instanciation de nouveaux objets
- Utilisation de l'opérateur this
- Création de classes imbriquées
- Accès aux classes imbriquées



## Hello, World Revisited

```
using System;

class Hello
{
    public static int Main( )
    {
        Console.WriteLine("Hello, World");
        return 0;
    }
}
```





## Définition de classes simples

- Données et méthodes ensemble dans une classe
- Les méthodes sont publiques, les données sont privées

```
class BankAccount
{
    public void Withdraw(decimal amount)
    { ... }
    public void Deposit(decimal amount)
    { ... }
    private decimal balance;
    private string name;
}
```


Les méthodes  
publiques décrivent  
les comportements



Les champs privés  
décrivent l'état  
inaccessible

## L'instanciation de nouveaux objets

- Déclarer une variable de classe ne crée pas un objet
  - Utilisez l'opérateur **new** pour créer un objet

```
class Program
{
    static void Main( )
    {
        Time now;
        now.hour = 11;
        BankAccount yours = new BankAccount( );
        yours.Deposit(999999M);
    }
}
```

now 

yours  →  new  
BankAccount  
object

## Utilisation de la clé de cette

- Le mot clé `this` fait référence à l'objet utilisé pour appeler la méthode
  - Utile lorsque les identificateurs de portées différentes s'affrontent
  - Optionnel dans les autres cas.

```
class BankAccount
{
    ...
    public void SetName(string name)
    {
        this.name = name;
    }
    private string name;
}
```

Si cette déclaration était  
`name = name ;`  
Que se passerait?

## Création de classes imbriquées

- Les classes peuvent être imbriquées dans d'autres classes

```
class Program
{
    static void Main( )
    {
        Bank.Account yours = new Bank.Account( );
    }
}
class Bank
{
    ... class Account { ... }
}
```

Le nom complet de la classe  
imbriquée comprend le nom  
de la classe externe

## Accès classes imbriquées

- Classes imbriquées peuvent aussi être déclarées comme public ou privé

```
class Bank
{
    public class Account { ... }
    private class AccountNumberGenerator { ... }
}
class Program
{
    static void Main( )
    {
        Bank.Account accessible; ✓
        Bank.AccountNumberGenerator inaccessible; ✗
    }
}
```

## Atelier 7.1: Création et utilisation de classes

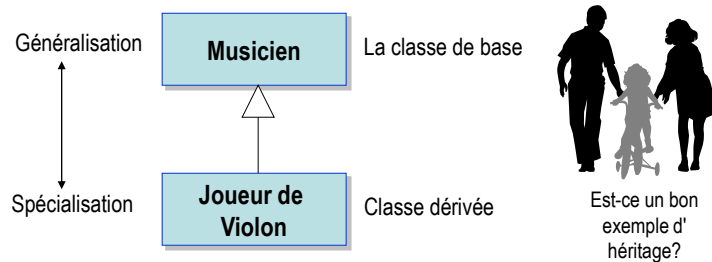


## Définition des systèmes orientés objet

- Héritage
- Les hiérarchies de classe
- Polymorphisme
- Classes de base abstraites
- Interfaces
- Liaison tardive et anticipée

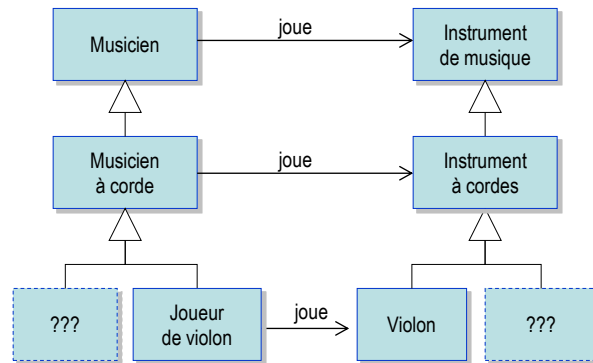
## Héritage

- L'héritage spécifie une relation *"est une sorte de"*
  - L'héritage est une relation de classe
  - De nouvelles classes spécialisent des classes existantes



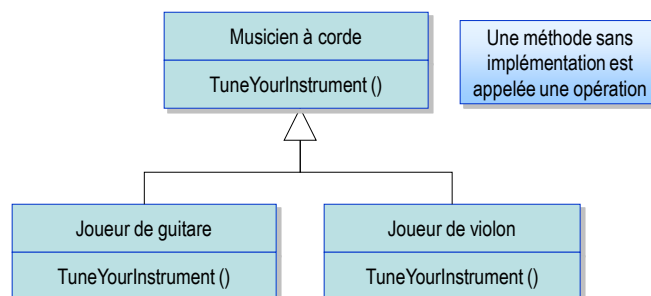
## Les hiérarchies de classe

- Les classes liées par des hiérarchies forme de classe héritage



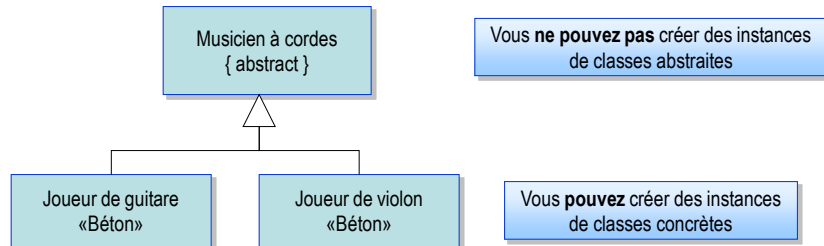
## Polymorphisme

- Le nom de la méthode réside dans la classe de base
- Les implémentations de la méthode résident dans les classes dérivées



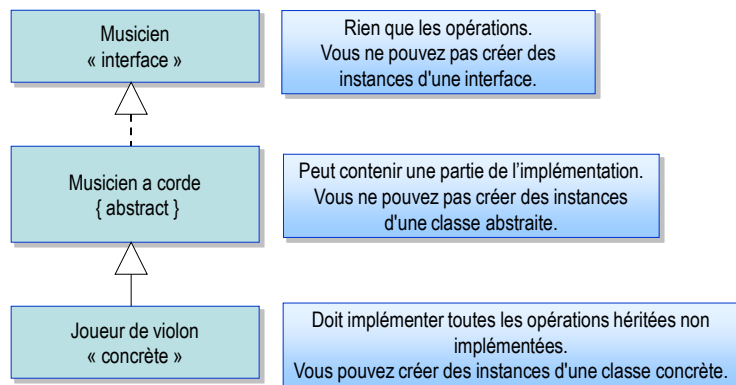
## Classes de base abstraites

- Certaines classes existent qu'uniquement pour être dérivée
  - Cela n'a aucun sens de créer des instances de ces classes
  - Ces classes sont *abstract*



## Interfaces

- Les interfaces contiennent uniquement des opérations, et non l'implémentation



## Liaison tardive et anticipée

- Les appels normaux de méthodes sont résolus à la compilation
- Les appels polymorphes de méthode sont résolus lors de l'exécution

