Orienté Objet
Les objets en C#

Les objets en C#

- Maintenant que nous avons vu comment définir un patron (des classes) pour créer des objets, il est temps de passer à l'étape suivante … à savoir instancier nos premiers objets en C#

Types primitifs vs Types références

- Le **C#** est un langage fortement typé !

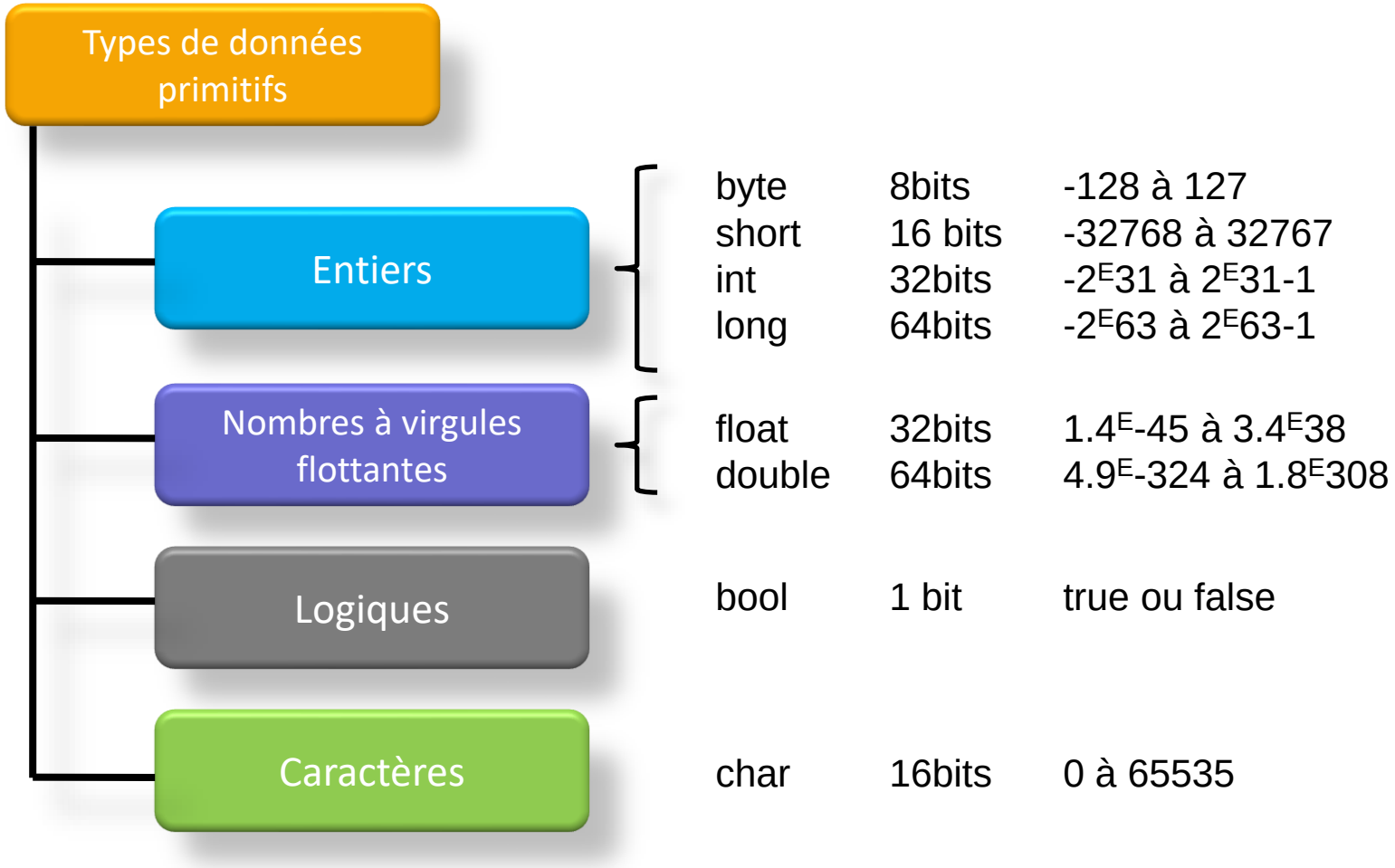
- Le type d'une donnée en C# précise les valeurs qu'elle peut contenir et les opérations que l'on peut réaliser dessus.

- Il existe deux types de données: **primitives et références**

  Primitive : contient **physiquement** la **valeur**.

  Référence : contient une **référence** vers la donnée en mémoire.

Types primitifs vs Types références



Types primitifs vs Types références

Types de données référence

Tous les types sauf les types primitifs

Exemple:

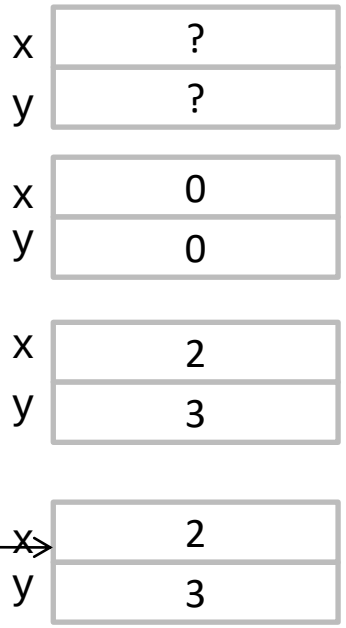
```
Point p;
p = new Point(2,3);
```

1 – recherche une place en mémoire

2 – assignation d'une valeur par défaut

3 – appel au constructeur de la classe

4 – création du pointeur



technocité
creative industries knowledge center

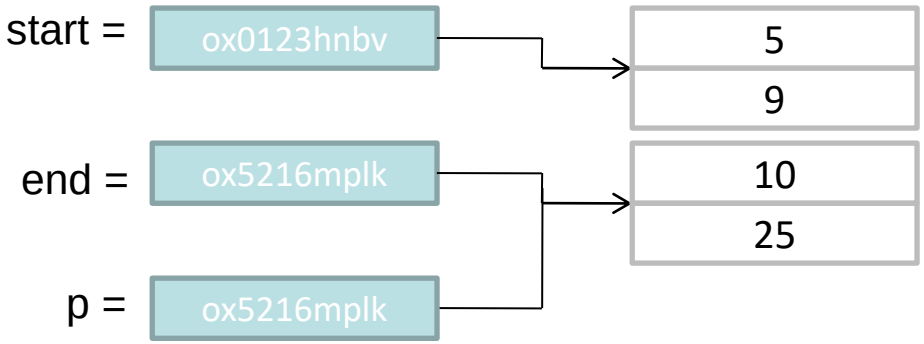
WAVENET
Intelligent IT Solutions

Types primitifs vs Types références

Que se passe-t-il si nous affectons un objet d'un certain type à un autre objet du même type ? Y a-t-il création d'un nouvel objet par copie ?

Exemple:

```
Point start = new Point(5,9);
Point end = new Point(10,25);
Point p = end;
```



Instancier un objet en C#

- En C# (et dans beaucoup d'autres langages orientés objet), il n'existe qu'un seul mot clé qui permet de créer un objet : **new**

```
MaClasse objetDeMaclasse = new MaClasse();
```

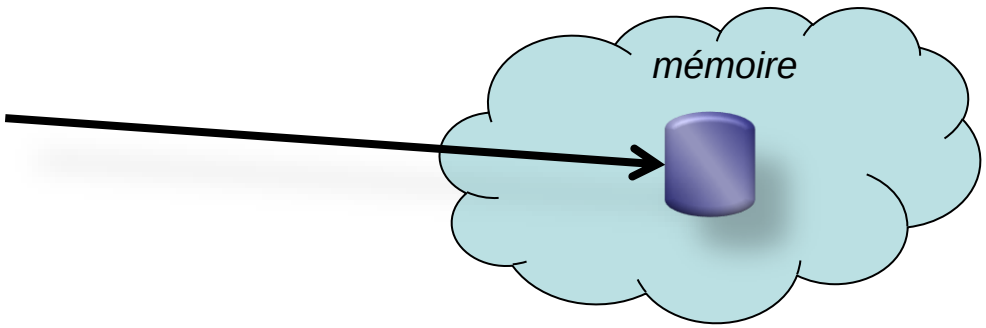
Appel au constructeur de la classe

Nom de la classe

Nom de l'objet

Création de l'objet en mémoire

objetDeMaclasse = référence vers l'objet **et non** l'objet en lui-même!



Le constructeur

- Si une classe peut être vue comme une usine servant à créer des objets, alors le **constructeur** de cette classe correspond aux chaînes de montage.

- Le constructeur d'une classe ressemble à s'y méprendre à n'importe quelle autre méthode de la classe si ce n'est que, un constructeur:

  - **Doit** avoir le **même** nom que la classe
  - **Doit** être invoqué si l'on désire instancier un objet de cette classe
  - N'a **jamais** de type de retour
  - En général, est **public**

Le constructeur

- Il existe toujours un constructeur par défaut si vous n'en définissez pas dans la classe. Il ne comporte aucun paramètre.

- Si vous définissez un constructeur avec paramètres, vous ne pouvez plus faire appel au constructeur par défaut à moins de le définir explicitement dans la classe.

```
class Velo
{
}
```

```
Velo unVelo = new Velo();
```

```
class Velo
{
   public Velo(string type){…}
}
```

~~`Velo unVelo = new Velo();`~~

```
class Velo
{
   public Velo(){…}
   public Velo(string type){…}
}
```

Surcharger le constructeur

- Une classe n'est pas limitée à posséder un seul constructeur.

- Pour une classe donnée, on va donc pouvoir créer autant de constructeurs que nécessaires à condition que ceux-ci aient tous une **signature** différente.

- Par signature, on entend la liste des paramètres que prend en entrée le constructeur.

```
class Velo
{
   public Velo(){…}
   public Velo(string type){…}
   public Velo(string type, string couleur){…}
   public Velo(Velo velo){…}
   …
}
```

Le desctructeur et garbage collector

- Les destructeurs ne peuvent être définis **que** dans des classes
- Une classe peut posséder **un seul** destructeur
- Les destructeurs ne peuvent pas être hérités ou surchargés
- Les destructeurs ne peuvent pas être appelés. Ils sont appelés automatiquement par le Garbage Collector
- Un destructeur n'accepte pas de modificateurs ni de paramètres
- Un destructeur porte le même nom que la classe précédé du ~

```
class Velo
{
   ~ Velo(){…}
}
```

Le desctructeur et garbage collector

- Le programmeur n'a aucun contrôle sur le moment où le destructeur est appelé car cela est déterminé par le garbage collector.

- Le garbage collector recherche les objets qui ne sont plus utilisés par l'application (c’est-à-dire des objets vers lesquels on n’a plus de référence).
  S'il considère qu'un objet est candidat à la destruction, il appelle le destructeur (s'il y a lieu) et libère la mémoire utilisée pour stocker l'objet. Les destructeurs sont également appelés à la fermeture du programme.

Attributs et méthodes d'une classe

Les attributs:

- Les attributs d'une classe doivent être déclarés en dehors de tout corps de méthode
- Les attributs peuvent être de type primitif ou référence
- Les attributs peuvent se voir attribuer les modificateurs d'accès suivants : **public**, **protected**, **private**, **internal** et **internal protected**
- Si aucun modificateur n'est spécifié alors l'attribut est considéré comme **private**
- Il n'est pas nécessaire d'attribuer une valeur par défaut aux attributs, ceux-ci prendront d'office une valeur par défaut en fonction de leur type

Attributs et méthodes d'une classe

Les méthodes:

- Les méthodes d'une classe permettent de réaliser des traitements sur les attributs de la classe, d'appeler d'autres méthodes (de la classe ou d'autres classes), de déclarer des variables locales (voir plus loin).
- Les méthodes peuvent se voir attribuer les modificateurs d'accès suivants : **public**, **protected**, **private**, **internal** ,**internal protected.**
- Si aucun modificateur n'est spécifié alors la méthode est considérée comme **private**
- Une méthode se distingue d'un attribut par sa signature:

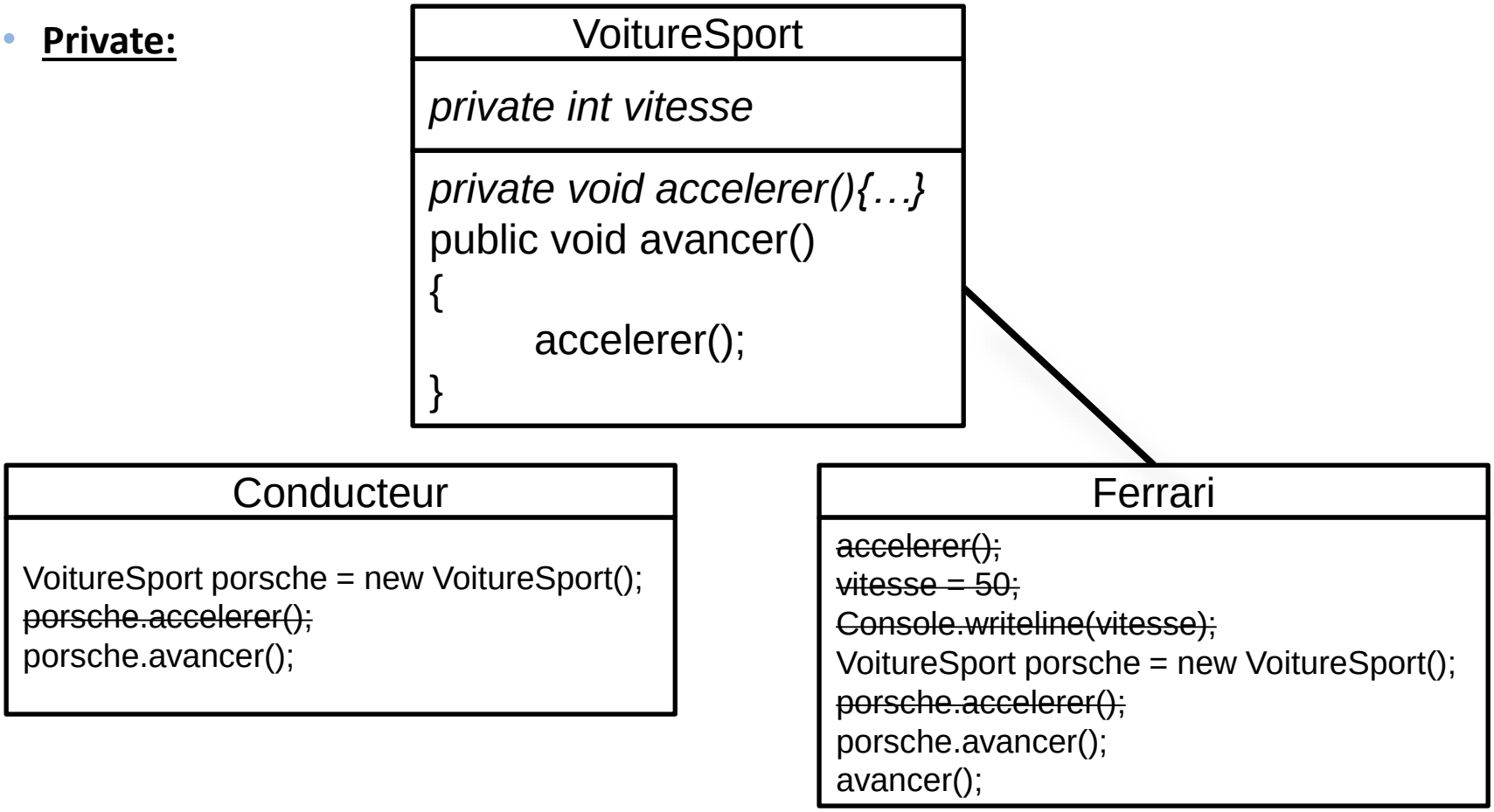
type de retour *nomDeLaMéthode ([**type paramètre1** nomParamètre1, **type paramètre2** nomParamètre2 ,…])*

{ instructions… }

- Le type de retour d'une méthode peut être de type primitif ou référence ou **void** si rien n'est retourné

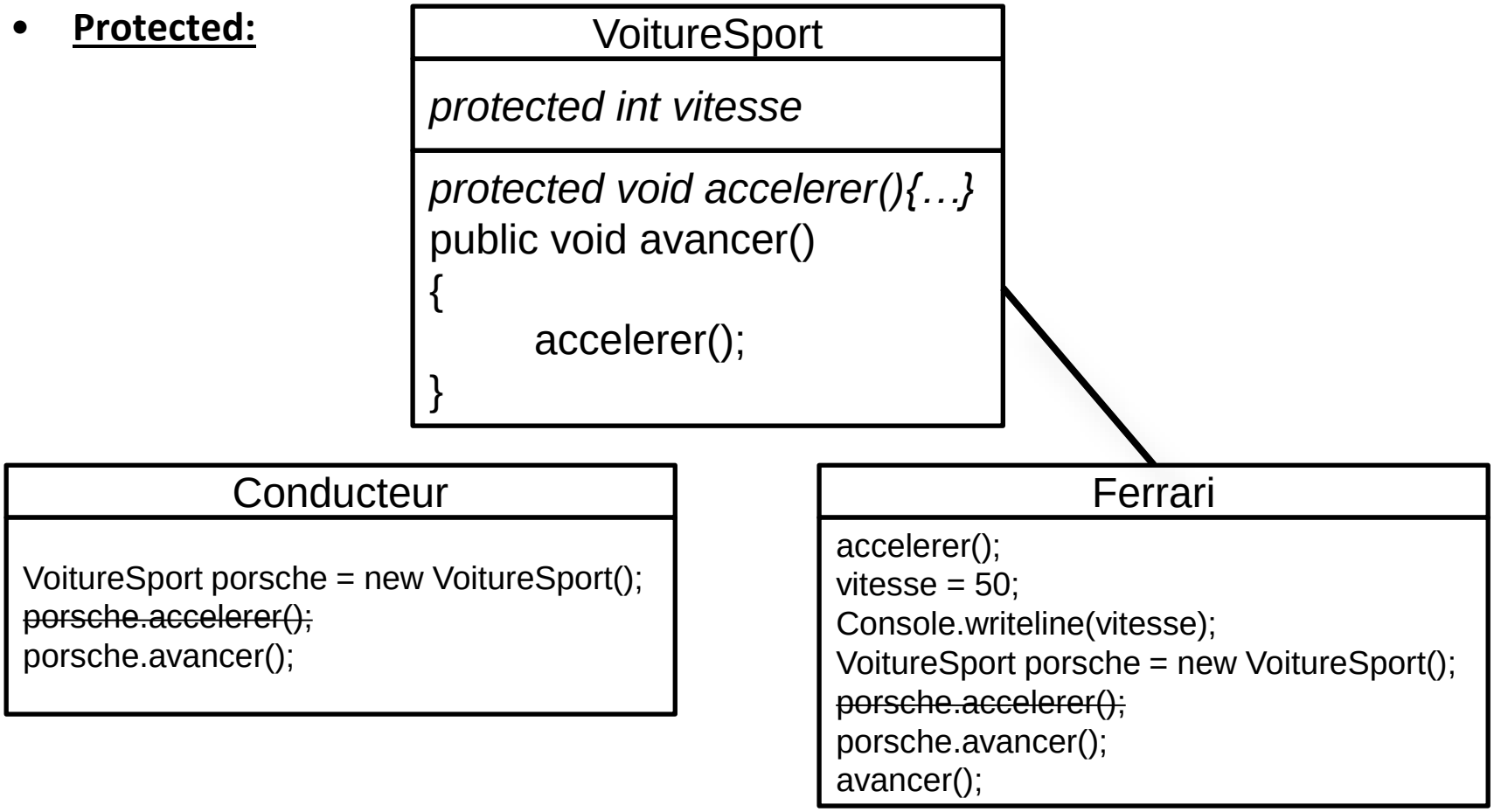
Impact des modificateurs d'accès sur les attributs et méthodes d'une classe

- **Private:**



Impact des modificateurs d'accès sur les attributs et méthodes d'une classe

- **Protected:**



Impact des modificateurs d'accès sur les attributs et méthodes d'une classe

- **Public:**

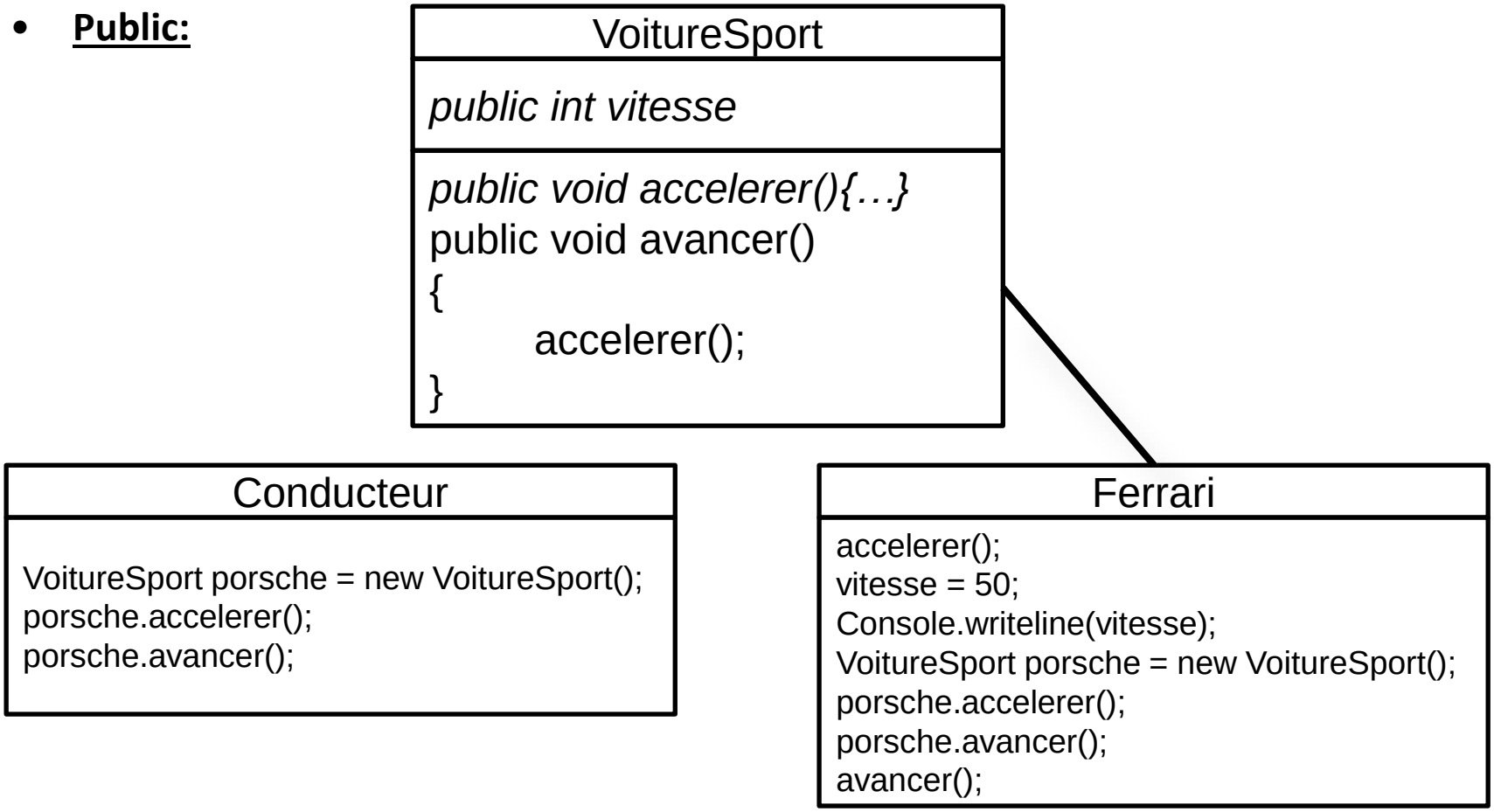


Tableau récapitulatif

Visibilité	Public	Protected	Internal protected	Internal	Private ou défaut
même classe	oui	oui	oui	oui	oui
autre classe ; même namespace	oui	non	oui	oui	non
sous-classe ; même namespace	oui	oui	oui	oui	non
sous-classe ; autre namespace	oui	oui (par héritage)	oui	non	non
autres classes ; autre namespace	oui	non	non	non	non

Variables locales

- Les variables locales sont des variables déclarée à l'intérieur des méthodes d'une classe (constructeur compris)
- Les variables locales n'ont de portée que dans le corps de la méthode où elles sont déclarées
- A l'inverse des attributs, les variables locales doivent être initialisées avant de pouvoir être utilisées

```
//a est inconnu

void maMéthode(param1, param2,…)
{
        int a = 3;
}

//a est inconnu
```

La récursivité

- La récursivité est la capacité qu'a une méthode à s'appeler elle-même

- Une méthode récursive doit toujours contenir une clause de « fin », pour ainsi éviter de faire des appels à l'infini. Ce type de fonction est notamment utilisée pour faire, par exemple, des recherches de fichiers sur le disque dur (dans les répertoires et sous-répertoires)

```
void maMéthode(param1, param2,…)
{
    maMéthode(param1, param2,…)
}
```

La récursivité - Exercice

- A l'aide du mécanisme de récursivité, écrire une classe qui permet de calculer la factorielle d'un nombre *n*

- La factorielle d'un nombre s'obtient avec la formule suivante :

$$n! = 1 * 2 * 3 * \ldots * (n-1) * n$$

- Exemple : $3! = 1 * 2 * 3 = 6$

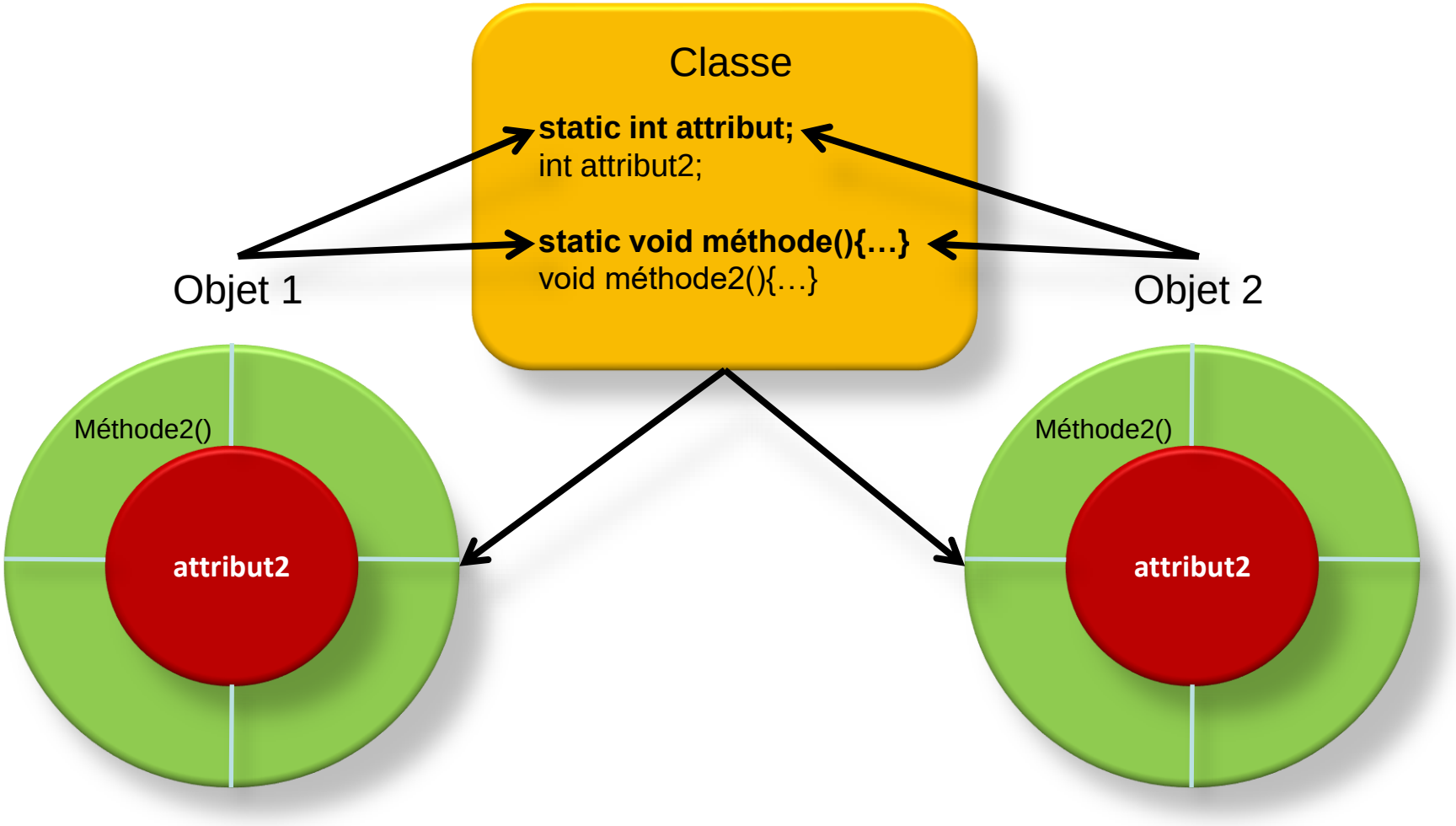
Variable d'instance vs Variable de classe
Méthode d'instance vs Méthodes de classe

- Un autre mot clé important et commun à beaucoup de langages orientés objet est le mot clé **static**.
- Ce mot clé sera utilisé sur les membres de la classe à savoir les attributs et méthodes de celle-ci.
- Lorsqu'un attribut ou une méthode est précédé du mot clé ***static*** , cela signifie que l'attribut ou la méthode n'est plus considéré comme appartenant à un objet en particulier mais à la classe en elle-même.

  On parle alors d'**attribut** ou de **méthode de classe** !

  … et tous les objets créés à partir de cette classe **partagent** entre eux les mêmes attributs et méthodes de classe
- Par opposition, un attribut ou une méthode sans mot clé ***static*** est un **attribut** ou une **méthode d'instance** et donc propre à l'objet.

Variable d'instance vs Variable de classe
Méthode d'instance vs Méthodes de classe



Variable d'instance vs Variable de classe
Méthode d'instance vs Méthodes de classe

- Pour utiliser un attribut ou une méthode de classe, il suffit d'utiliser le nom de la classe suivi de l'opérateur et du nom de l'attribut ou de la méthode :

```
class Velo
{
   static string couleur
   static string Tourner(){….}
}
```

```
Velo.couleur;
Velo.Tourner();
```

- Une règle cependant à retenir : ce qui est statique ne peut faire référence qu'à du statique => une méthode statique ne peut faire référence qu'à un attribut statique ou à une autre méthode statique. Seul le constructeur peut faire référence à des attributs et méthodes statiques sans être déclaré comme statique.

Variable d'instance vs Variable de classe
Méthode d'instance vs Méthodes de classe - Exercice

- Créer une classe ***boulangerie*** qui contient une méthode ***main()*** dans laquelle on crée un certain nombre d'objets de type *[insérer ici votre pâtisserie préférée ;-)]*

- Le nombre d'objets à créer doit être passé en **argument** au programme lors de son exécution.

- A la fin de la journée de travail, on doit pouvoir être capable de dire combien de pâtisseries la boulangerie a créées…

Constructeurs statiques

- Les constructeurs de classe (qui sont des méthodes à part entières avec un rôle bien spécifique) peuvent être déclarés statiques.
- L'avantage de déclarer un constructeur comme étant statique est de pouvoir initialiser des attributs statiques de la classe sans avoir besoin de créer une instance de celle-ci.
- Une classe ne peut avoir qu'un seul constructeur statique.

```
class Velo
{
   static string couleur;
   static velo()
  {
    couleur = "Rouge";
  }
}
```

Constructeurs statiques - Exercice

- Créer une classe qui déclare un constructeur statique.
- La classe doit disposer d'une méthode qui retourne le nombre qu'on lui passe en paramètres en lettre.
- Pour connaître le mot, cette méthode consulte un tableau qui contient les libellé des nombres de 1 à 10
- Ce tableau est initialisé par le constructeur
- Il ne faut pas créer une instance de la classe
- Créer une classe avec une méthode Main qui invoque cette méthode et affiche le résultat.