Les personnages rencontrent le personnage courant

```
public static void main(String[] args)
{
   Personnage   lePersonnage = new Personnage(...);
   Personnage[] personnages  = new Personnage[3];

   personnages[0] = new Voleur(...); // Correct ?
   personnages[1] = new Guerrier(...);
   personnages[2] = new Sorcier(...);

   for (int i = 0; i < personnages.length; ++i)
   {
      personnages[i].rencontrer(lePersonnage);
   }
}
```

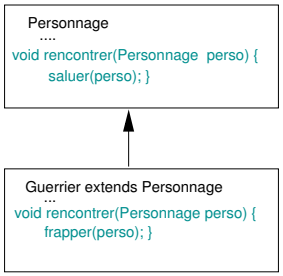
☞ Peut-on mettre un `Sorcier`, un `Voleur` ou un `Guerrier` dans un tableau de `Personnage` ?

Héritage du type : rappel

Dans une hiérarchie de classes :

- Un objet d'une sous-classe hérite le type de sa super-classe
- L'héritage est transitif
- Un objet peut donc avoir **plusieurs types**

Choix de la méthode à exécuter (1)



Que fait le code suivant :

```
// ...
Personnage unPersonnage = new Guerrier(...);
unPersonnage.rencontrer(unAutrePersonnage);
```

☞ Quelle méthode `rencontrer(Personnage)` va être exécutée ?

Choix de la méthode à exécuter (2)

```
// ...
Personnage unPersonnage = new Guerrier(...);
unPersonnage.rencontrer(unAutrePersonnage);
```

1. Résolution *statique* des liens :
   - Le type apparent (type de la variable) est déterminant
   - `unPersonnage` est *déclarée* comme une *variable* de type `Personnage`
   - Choix de la méthode de la classe `Personnage` (le personnage salue le personnage courant !)
2. Résolution *dynamique* des liens :
   - Le type effectif (celui de l'objet effectivement stocké dans la variable) est déterminant
   - `unPersonnage` contient la référence à un objet de type `Guerrier`

Résolution dynamique des liens

Java met en œuvre le principe de « **résolution dynamique des liens** »

☞ C'est le type effectif et non le type apparent qui est pris en compte

Résolution dynamique des liens – Exemple (1)

```
class Jeu {
  private Personnage joueur;
  private Personnage[] adversaires;
  // ..
  public void tourDeJeu() {
    for (int i = 0; i < adversaires.length; ++i)
    {
      adversaires[i].rencontrer(joueur);
    }
  }
  // ...
}
```

Que se passe-t-il si :

```
adversaires[0] = new Sorcier(...);
adversaires[1] = new Guerrier(...);
// ...
leJeu.tourDeJeu();
```

Résolution dynamique des liens – Exemple (2)

- Avec la résolution « statique » des liens, dans `tourDeJeu`, ce serait toujours `rencontrer(Personnage)` de `Personnage` qui serait appelé (**c'est le type apparent des variables qui décide**)
  - ☞ Le personnage principal du jeu se fait saluer deux fois : une fois par le guerrier et une autre fois par le sorcier
- Avec la résolution « dynamique » des liens, dans `tourDeJeu`, `rencontrer(Personnage)` de `Personnage` est appelée pour le sorcier mais `rencontrer(Personnage)` de `Guerrier` est appelée pour le guerrier (**c'est le type effectif qui décide**)
  - ☞ Le personnage principal du jeu se fait saluer par le sorcier … mais frapper par le guerrier !
  - ☞ C'est ce qui va se passer en Java

Polymorphisme

Les deux ingrédients :

- héritage du type dans une hiérarchie de classes,
- et résolution dynamique des liens

permettent de mettre en œuvre ce que l'on appelle le **polymorphisme**.

- Un même code s'exécute de façon différente selon la donnée à laquelle il s'applique.

☞ Nous y reviendrons plus en détail au cours prochain