Les `Guerrier` font bande à part

- Pour un personnage non-`Guerrier`:

```
public void rencontrer(Personnage unPersonnage) { saluer(lePersonnage); }
```

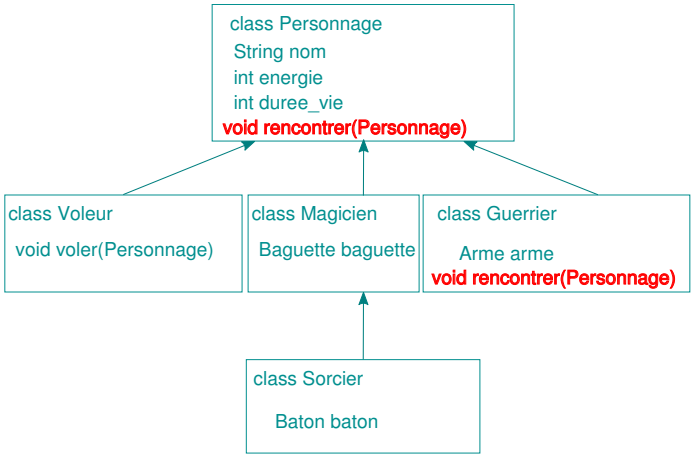
- Pour un `Guerrier`

```
public void rencontrer(Personnage lePauvre) { frapper(lePauvre); }
```

Faut-il re-concevoir toute la hiérarchie ?

☞ Non, on ajoute simplement une méthode `rencontrer(Personnage)` spéciale dans la sous-classe `Guerrier`

Les `Guerrier` font bande à part : masquage/redéfinition

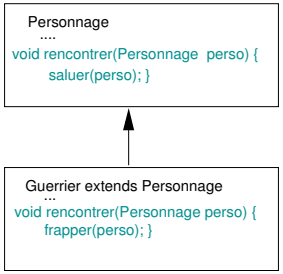


Masquage/Redéfinition dans une hiérarchie

Masquage : pour les variables (« *shadowing* »)
Redéfinition : pour les méthodes (« *overriding* »)

- Masquage : un identificateur qui en cache un autre
- Redéfinition : une méthode déjà définie dans une super-classe a une nouvelle définition dans une sous-classe
- Situations possibles dans une hiérarchie :
  - Même nom d'attribut ou de méthode utilisé sur plusieurs niveaux
  - Peu courant pour les attributs
  - Très courant et **pratique** pour les méthodes

Masquage /Redéfinition dans une hiérarchie (2)



La méthode `rencontrer` de `Guerrier` **redéfinit** celle de `Personnage`

- Un objet de type `Guerrier` n'utilisera donc **jamais** la méthode `rencontrer` de la classe `Personnage`
- Vocabulaire OO :
  - Méthode héritée = méthode générale, *méthode par défaut*
  - Méthode qui redéfinit la méthode héritée = *méthode spécialisée*

Accès à une méthode masquée

- Il est parfois souhaitable d'accéder à une méthode/un attribut masqué(e)
- Exemple :
  - Le `Guerrier` commence par rencontrer le personnage comme le fait n'importe quel personnage (il le salue) avant de le frapper !
- Code désiré :
  1. Personnage non-`Guerrier` :
     - Méthode générale (`rencontrer` de `Personnage`)
  2. Personnage `Guerrier` :
     - Méthode spécialisée (`rencontrer` de `Guerrier`)
     - Appel à la méthode générale depuis la méthode spécialisée

Accès à une méthode masquée (2)

Pour accéder aux attributs masqués et aux méthodes redéfinies de la super-classe :

- on utilise le mot réservé `super`
- Syntaxe : `super.` *méthode* ou *attribut*
- Exemple :

```
class Guerrier extends Personnage  {
  //...
  public void rencontrer (Personnage perso) {
    super.rencontrer(perso); // salutation d'usage !!
    frapper(perso);
  }
}
```