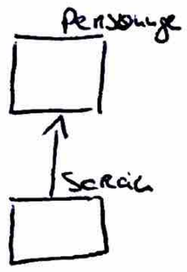
POLYMORPHISME

UN MEME CODE S'ADAPTE AU TYPE DE DONNÉES A L'AQUELLE ELLE S'APPLIQUE

```
Personnage p = new Sorcier(...);
```

⇓ OUI ! POSSIBLE



UN OBJET DE TYPE SORCIER EST AUSSI UN OBJET DE TYPE PERSONNAGE

UN OBJET PEUT ETRE DE PLUSIEURS TYPE DE PART LA TRANSITIVITÉ DE L'HERITAGE

RESOLUTION STATIQUE DES LIENS

TYPE APPARENT

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

LE TYPE DE LA VARIABLE EST DETERMINANT DANS CETTE RESOLUTION ici → Personnage

APPEL DE LA METHODE DE LA CLASSE PERSONNAGE

=> ON DIT STATIQUE CAR ON N'A PAS BESOIN D'ATTENDRE L'EXECUTION DU PROGRAMME POUR FAIRE LE CHOIX DE LA METHODE A APPLIQUER

RESOLUTION DYNAMIQUE DES LIENS (DANA)

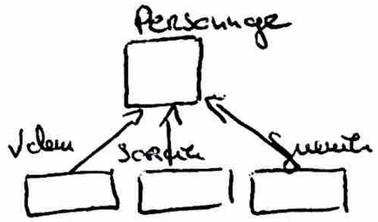
TYPE EFFECTIF

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

LE TYPE EST DETERMINANT

APPEL DE LA METHODE DE LA CLASSE GUERRIER

EXEMPLE TABLEAU DE PERSONNAGES.




```
Personnage[] personnages = new Personnage[3]
personnages[] = new Voleur(..);
personnages[] = new Guerrier(..);
personnages[] = new Sorcier(..);
```

↑ TYPE 1 SUPER CLASSE

↑ TYPE 2 CLASSE HERITÉE

DEFINITION

INSTANCES D'UNE SOUS-CLASSE GARDANT SES PROPRIETES PROPRES

⇓

1. EN ARGUMENTS D'UNE METHODE
2. LORS D'AFFECTATION

⇓

CHOIX DES METHODES A INVOQUER SUPER CLASSE OU CLASSE

⇓

LORS DE L'EXECUTION DU PROGRAMME NATURE REELLE DES INSTANCES

RESOLUTION DYNAMIQUES DES LIENS (DANA)

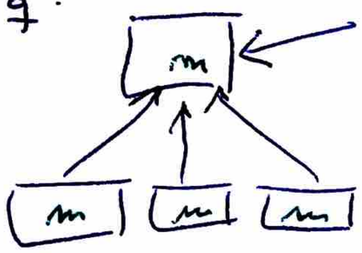
EXEMPLE

```
class Personnage {
  public
  void rencontrer(Personnage p){
    System.out.print("Bonjour");
  }
}

class Guerrier {
  public
  void rencontrer(Personnage p){
    System.out.print("Baam")
  }

class Rencontre {
  public static void main(...){
    Guerrier g = new Guerrier(...);
    Voleur v = new Voleur(...);
    uneRencontre(g, v)
  }
  static void uneRencontre(Personnage a
    Personnage b ){
      Syso(a.getNom());
      Syso("rencontre");
      Syso(b.getNom()+" ; ");
    a.rencontrer(b);
  }
}
```

RESOLUTION DYNAMIQUE

METHODES ABSTRAITES

Pq ?



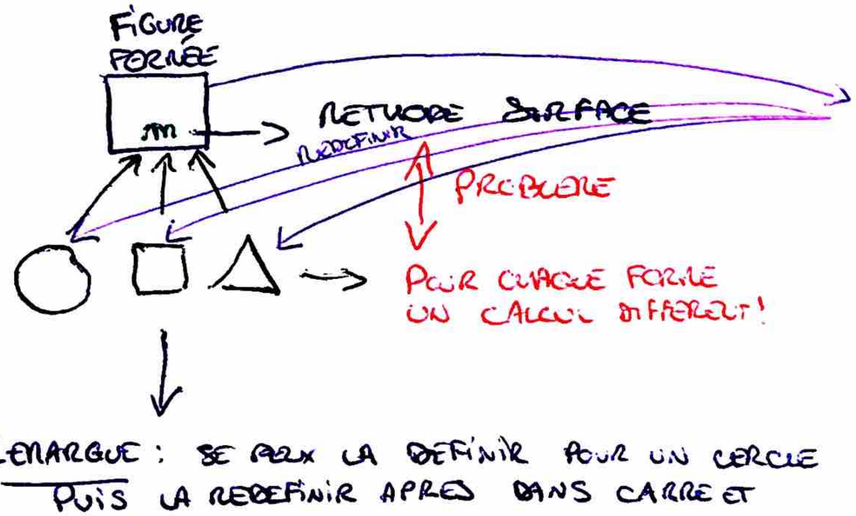
DEFINITION GENERALE D'UNE METHODE ? COMPATIBLE AVEC TOUTES SOUS-CLASSES ?

SYNTAXE

Typeretour

```
abstract class ...
  abstract Type NomMethode ( ... );
```

⚠ RÈGLE : CLASSE ABSTRAITE NON INSTANCIABLE




```
public abstract double surface();
```

⚠ ON PEUT UTILISER LA METHODE ABSTRAITE DANS LE CORPS D'UNE METHODE NON ABSTRAITE

```
public double volume (double hauteur) {
    return hauteur * surface()
}
```

REMARQUE : SE PEUX LA DEFINIR POUR UN CERCLE PUIS LA REDEFINIR APRES DANS CARRE ET TRIANGLE...

↓ NON !

SOURCE D'ERREURS

↓ SOLUTION

DECLARER LA METHODE ABSTRAITE

=

= SIGNALER QU'UNE METHODE DOIT EXISTER DS CHQUE SES CLASSE

= NE PAS LA DEFINIR DANS LA SUPER-CLASSE

⚠ LA CLASSE DES METHODES ABSTRAITES DOIT AUSSI ETRE ABSTRAITE !

⚠ SOUS CLASSE UTILISANT METHODE ABSTRAITE RESTE ABSTRAITE TANT QUE METHODES NONT PAS ETE TOUTES REDEFINIES.

POLYMORPHISME COMPLÉMENTS

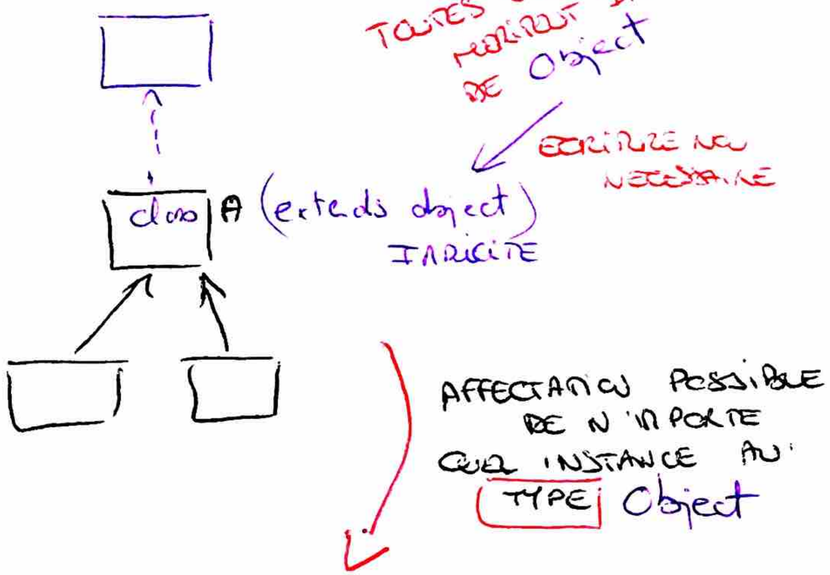
CONSTRUCTEURS & POLYMORPHISME

RÈGLE : TOUT CONSTRUCTEUR DE SOUS-CLASSE INVOQUE UN CONSTRUCTEUR DE LA SUPER-CLASSE QUAND PAS D'APPEL EXPLICITE À UN CONSTRUCTEUR VIA super()

→ IL Y A APPEL IMPLICITE AU CONSTRUCTEUR PAR DÉFAUT.

⇓

NE PAS INVOQUER DE METHODE POLYMORPHIQUE DANS CORPS D'UN CONSTRUCTEUR.



Object v = new A(...);

contient
- → toString → Faire un toString d'un objet de type Classe ou affiche sa référence ! X401...
- → equals → Comparaison == des références (si non redéfini)
- → clone

TRANSTYPAGE

CONVERTIR UN OBJET D'UN TYPE EN UN AUTRE TYPE

ex : (Rectangle) Carré