La Programmation Orientée Objet (POO) avec Java

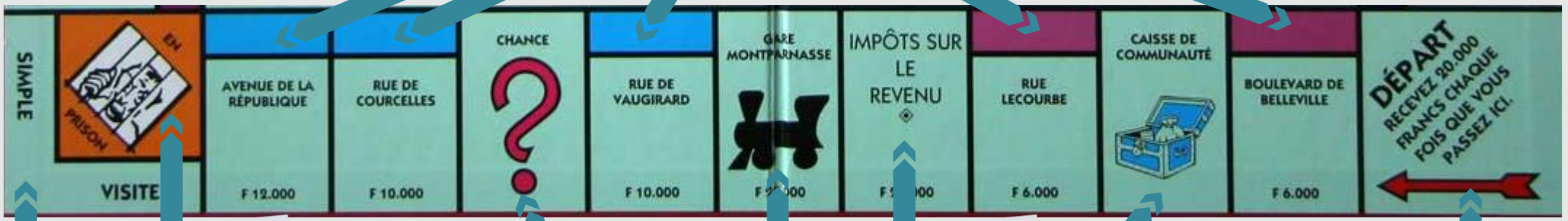
Module 5 – L'héritage

Objectifs

- Découvrir le concept de l'héritage
- Comprendre l'intérêt de l'héritage
- Savoir mettre en place de l'héritage et ses concepts associés

L'héritage

Les cases du plateau de jeu du Monopoly



Un terrain

Une case « Rien »

La prison

Une gare

Une case « Payer »

Une case « Piocher »

La case départ

L'héritage

Des classes avec du code dupliqué

Case

-nom:String

+Case(nom:String)
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String

AllezEnPrison

-nom:String

+AllezEnPrison()
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String

Gare

-nom:String
-prixAchat:int
-txComplGroupe:int
-hypothequee:boolean

+Gare(nom:String)
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String
-payerLoyer(j:Joueur, proprio:joueur)
-calculerCompletionGroupe(j:Joueur)
+getGroupe():Groupe
+getCompletionGroupe():int
+getProprio():Joueur
+setProprio(j:Joueur)
+isHypothequee():boolean
+setHypothequee()
+getValeur():int

ServicePublic

-nom:String
-prixAchat:int
-txComplGroupe:int
-hypothequee:boolean

+ServicePublic(nom:String)
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String
-payerLoyer(j:Joueur, proprio:joueur)
-calculerCompletionGroupe(j:Joueur)
+getGroupe():Groupe
+getCompletionGroupe():int
+getProprio():Joueur
+setProprio(j:Joueur)
+isHypothequee():boolean
+setHypothequee()
+getValeur():int

Terrain

-nom:String
-prixAchat:int
-txComplGroupe:int
-hypothequee:boolean
-loyer:int[]
-coutConstruction:int
-niveauConstruction:int
-nbMaisonsDispo:int
-nbHotelsDispo:int

+Terrain(nom:String, prixAchat:int, groupe:Groupe, coutConstruction:int)
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String
-payerLoyer(j:Joueur, proprio:joueur)
-calculerCompletionGroupe(j:Joueur)
+getGroupe():Groupe
+getCompletionGroupe():int
+getProprio():Joueur
+setProprio(j:Joueur)
+isHypothequee():boolean
+setHypothequee()
+getValeur():int
+getCoutConstruction():int
+getNiveauConstruction():int
+getNbMaisonsDispo():int
+getNbHotelsDispo():int
+construire()

Depart

-nom:String

+Depart()
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String
-payerSalaire(j:Joueur)

ImpotTaxe

-nom:String
-prix:int

+ImpotTaxe(nom:String, prix:int)
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String

Prison

-nom:String
-nbTentatives:Map<Joueur,Integer>

+Prison()
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String

CaisseDeLaCommunaute

-nom:String
-cartes:Pioche

+CaisseDeLaComunaute()
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String
-tirerCarte(j:Joueur)
+remettreEnJeu(Action carte)

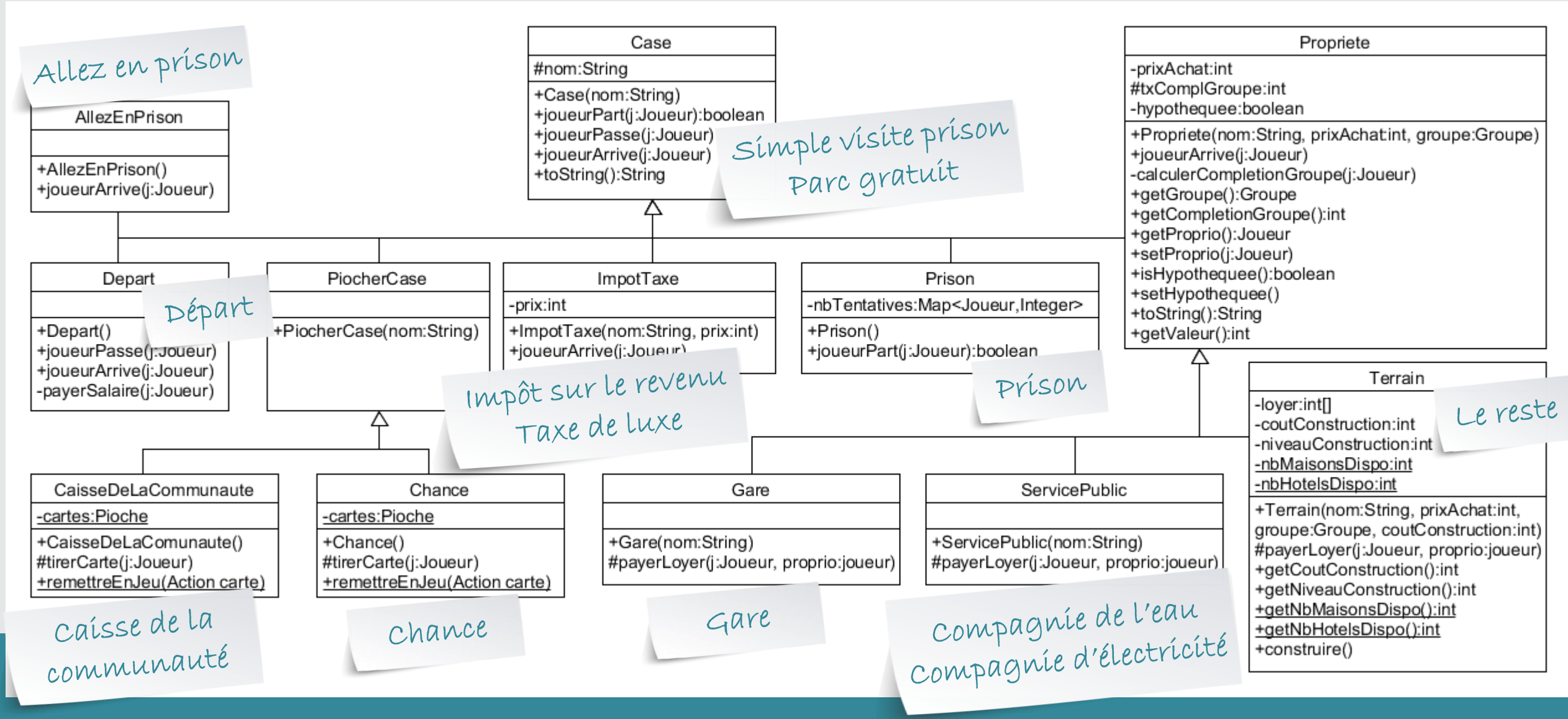
Chance

-nom:String
-cartes:Pioche

+Chance()
+joueurPart(j:Joueur):boolean
+joueurPasse(j:Joueur)
+joueurArrive(j:Joueur)
+toString():String
-tirerCarte(j:Joueur)
+remettreEnJeu(Action carte)

L'héritage

La solution : mutualisation du code par héritage



La classe parent

```
public class Case {
  protected String nom;

  public Case(String nom) {
    this.nom = nom;
  }
  public boolean joueurPart(Joueur j) {
    System.out.printf("%s est sur la case %s%n", j, this.nom);
    return true;
  }
  public void joueurPasse(Joueur j) {}
  public void joueurArrive(Joueur j) {
    System.out.printf("%s arrive sur la case %s%n", j, this.nom);
  }
}
```

visibilité
protected

L'héritage

Une classe enfant

extends est le mot clef permettant d'indiquer l'héritage

```
public class ImpotTaxe extends Case {

  private int prix;

  public ImpotTaxe(String nom, int prix) {
    super(nom);
    this.prix = prix;
  }

  @Override
  public void joueurArrive(Joueur j) {
    super.joueurArrive(j);
    System.out.printf("%s paye %d€ à la banque%n", j, this.prix);
    j.debiter(this.prix);
  }
}
```

Ajout d'un attribut supplémentaire en plus de celui défini dans Case

super() permet de faire appel à l'un des constructeurs de la classe parent

L'annotation @Override indique la substitution de la méthode

La méthode **joueurArrive()** est substituée : la version de cette méthode définie dans une classe parent est remplacée par celle-ci

L'héritage

Une autre classe enfant

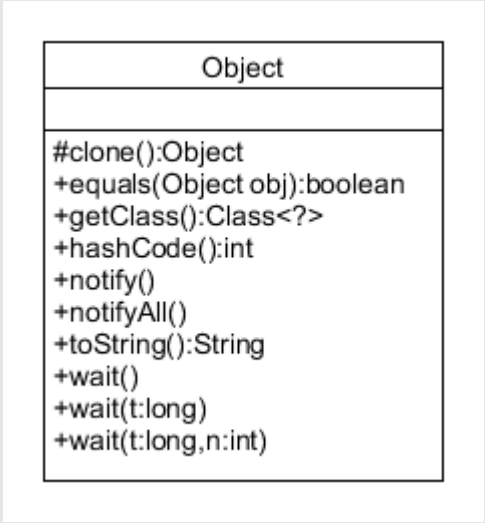
Le mot clef **super.** permet de faire appel à la méthode telle qu'elle était avant la substitution

```
public class Depart extends Case {
  private static final int SALAIRE = 200;

  public Depart() {
    super("Départ");
  }

  @Override
  public void joueurPasse(Joueur j) {
    payerSalaire(j);
  }

  @Override
  public void joueurArrive(Joueur j) {
    super.joueurArrive(j);
    payerSalaire(j);
  }

  private void payerSalaire(Joueur j) {
    System.out.printf("%s touche %d€%n", j, SALAIRE);
    j.crediter(SALAIRE);
  }
}
```

L'héritage

La classe Object

- Toute classe sans héritage explicite hérite implicitement de la classe Object
  - Donc directement ou indirectement, toute classe hérite de Object
  - Donc dans toute classe, appel possible aux méthodes publiques définies dans Object
  - Donc dans toute classe, il est possible de substituer les méthodes publiques et protégées de Object



L'héritage

La substitution de la méthode toString()

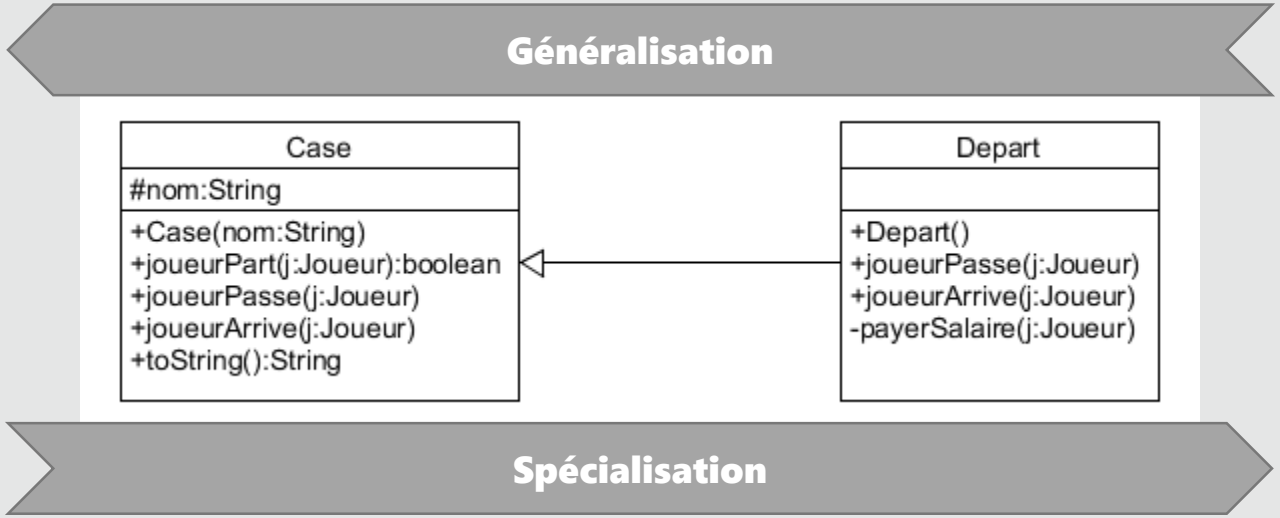
```java
public class Case {
  protected String nom;

      …

  @Override
  public String toString() {
    return this.nom;
  }
}
```

```java
public class Propriete extends Case {

  private int prixAchat;
  private Joueur proprio;
  private Groupe groupe;
  protected int txComplGroupe;
  private boolean hypothequee;

      …

  @Override
  public String toString() {
    return String.format("%s (%s)", super.toString(), this.groupe);
}
```

L'héritage

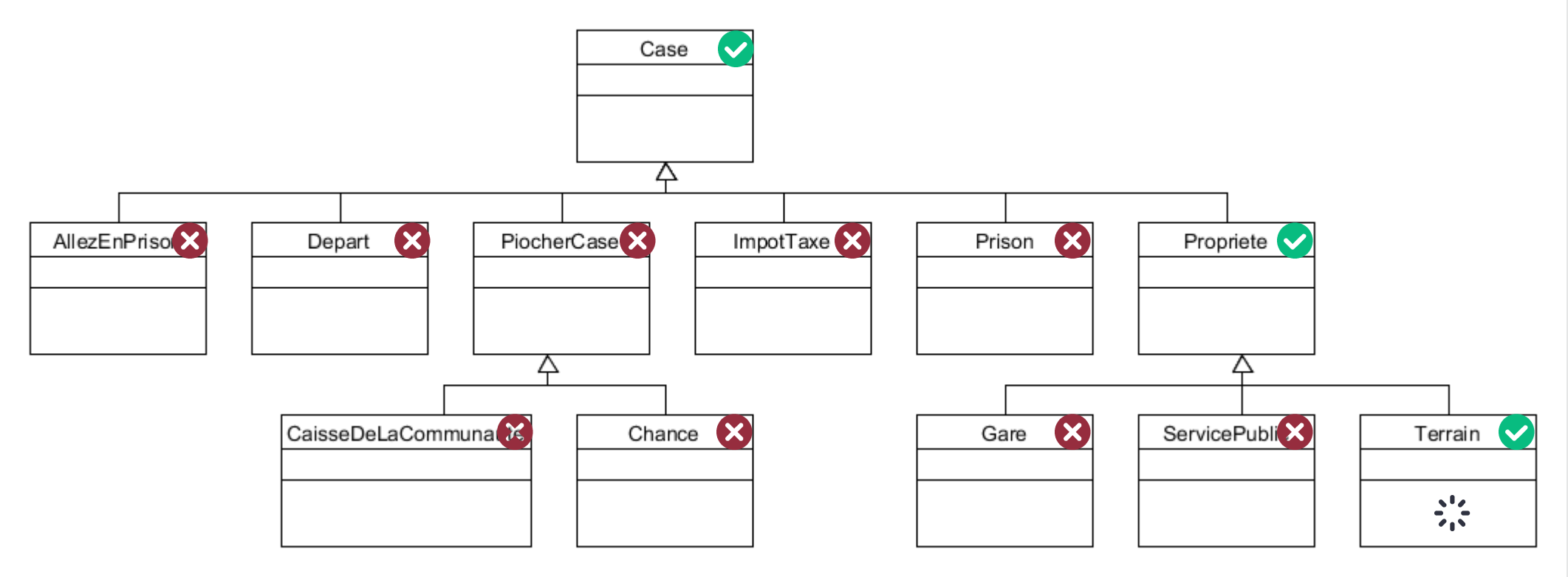
Généralisation et spécialisation

- Pour savoir si un héritage entre deux classes est possible
  - Utilisation de « est un cas particulier de »
    - Exemples :
      - Joueur est un cas particulier de Propriété ✗ — Joueur n'hérite pas de Propriété
      - Départ est un cas particulier de case ✓ — Départ hérite de Case



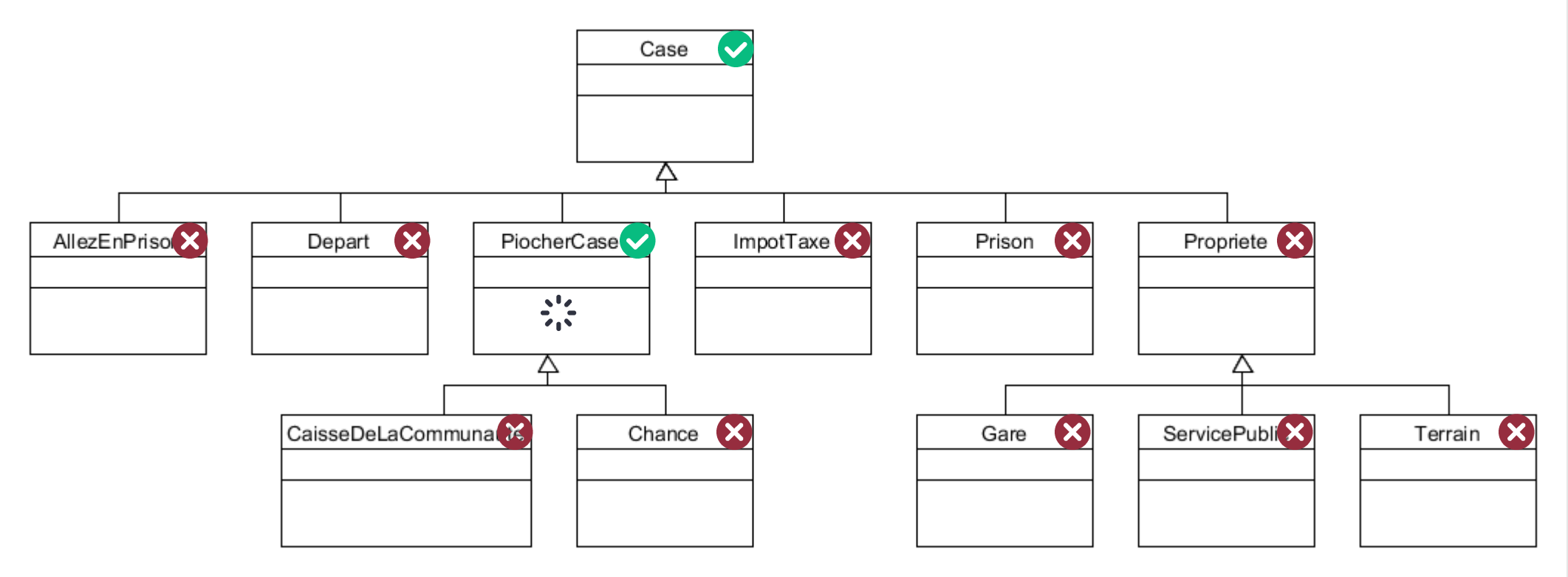
L'héritage

Le transtypage



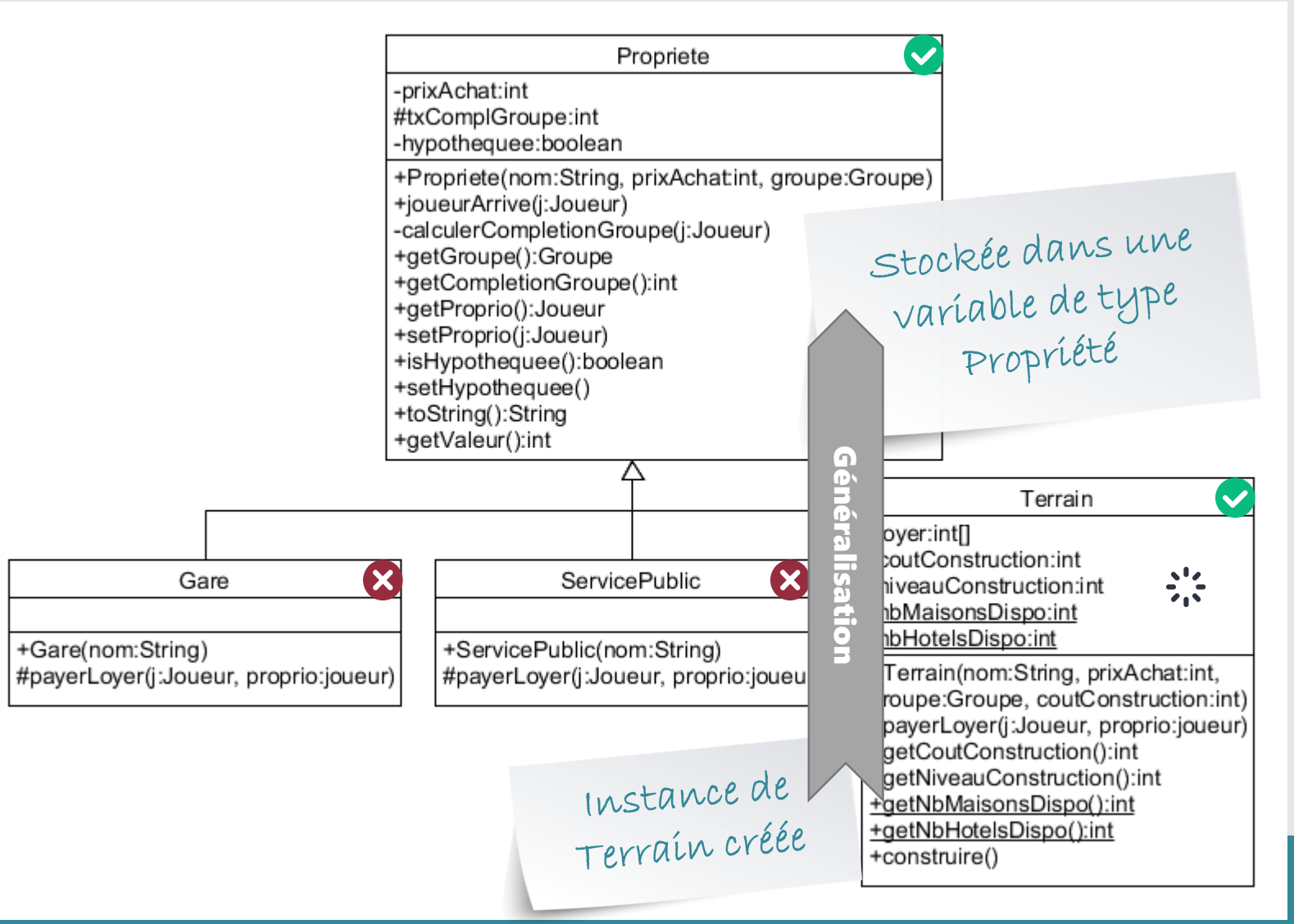
L'héritage

Le transtypage



L'héritage

Le transtypage ascendant



- Une instance peut être référencée par une variable d'un type ascendant au type réel de l'instance

Le transtypage ascendant

- Une instance peut être considérée en tant qu'instance de l'une de ses classes ancêtres

```
void ajouterPropriete(Propriete p)
```

belleville : Terrain	
prixAchat	60
txComplGroupe	0
Hypotheque	false
groupe	

montparnasse : Gare	
prixAchat	200
txComplGroupe	0
Hypotheque	false
groupe	

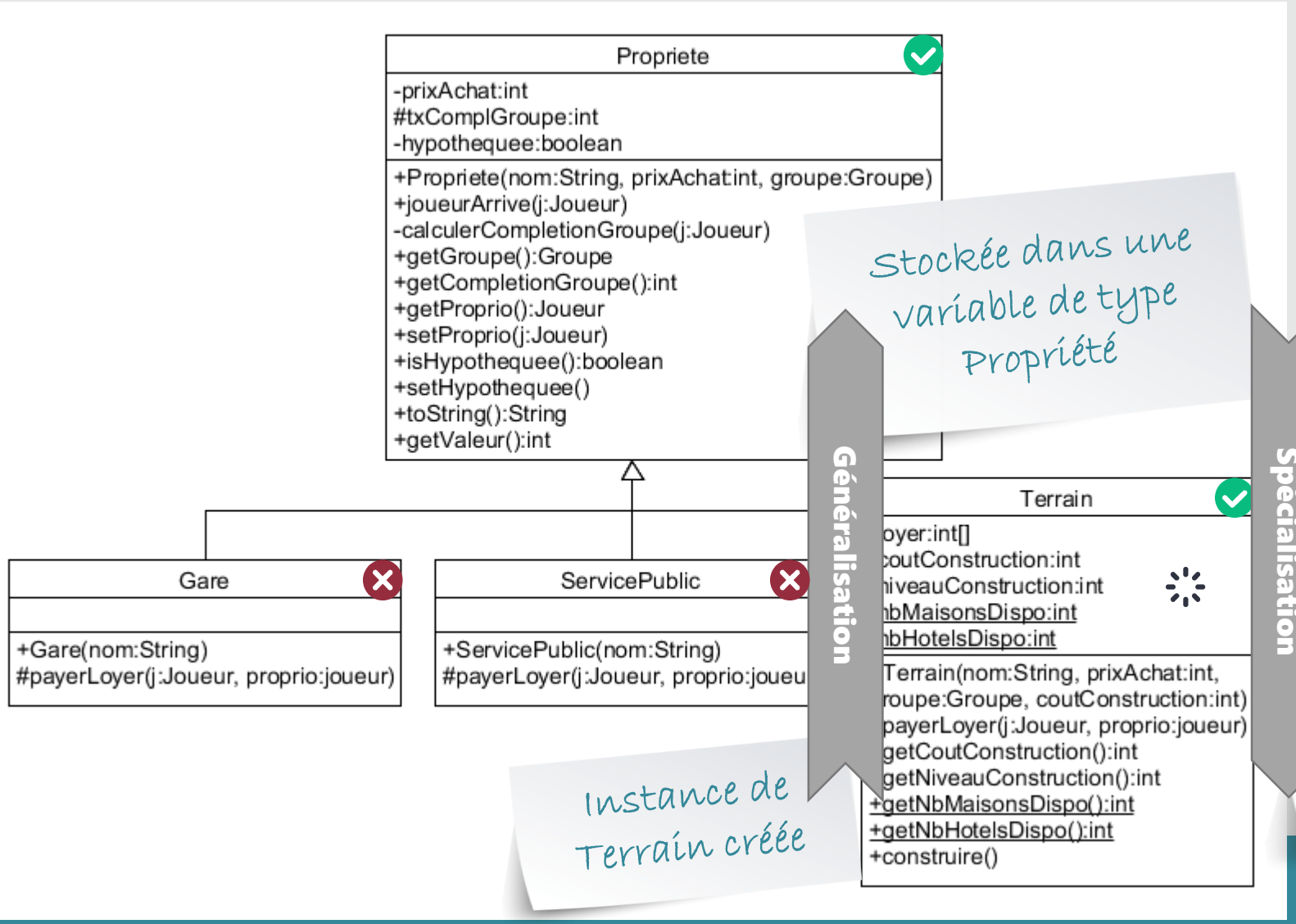
eau : ServicePublic	
prixAchat	200
txComplGroupe	0
Hypotheque	false
groupe	

```
ajouterPropriete(belleville); ajouterPropriete(montparnasse); ajouterPropriete(eau);
```

Aucun mot clef nécessaire

L'héritage

Le transtypage descendant



- Une instance peut à nouveau être référencée par une variable du type réel de cette instance

Le transtypage descendant

- Exemple :

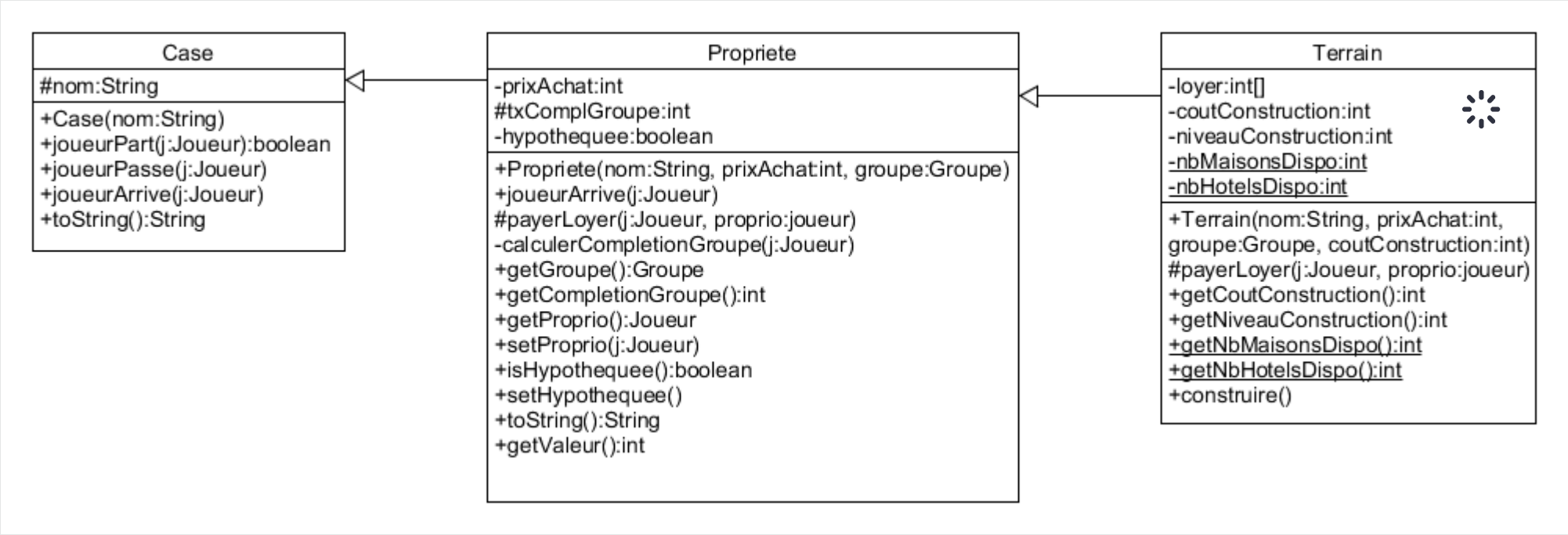
```
Propriete[] proprietes = groupe.getProprietes();
Propriete pr = proprietes[0];
// seuls les terrains sont constructibles et
// il faut que le joueur soit le propriétaire de tout le groupe
if (pr instanceof Terrain && this.equals(pr.getProprio()) && pr.getTxComplGroupe() == 100) {
  int nbConstructions = 0;
  for (Propriete p : proprietes) {
    Terrain t = (Terrain) p;
    nbConstructions += t.getNiveauConstruction();
  }
    …
```

`instanceof` est le mot clef permettant de tester le type de l'instance

`(Terrain) p` permet d'effectuer un « cast » de l'instance p en tant qu'instance de type Terrain

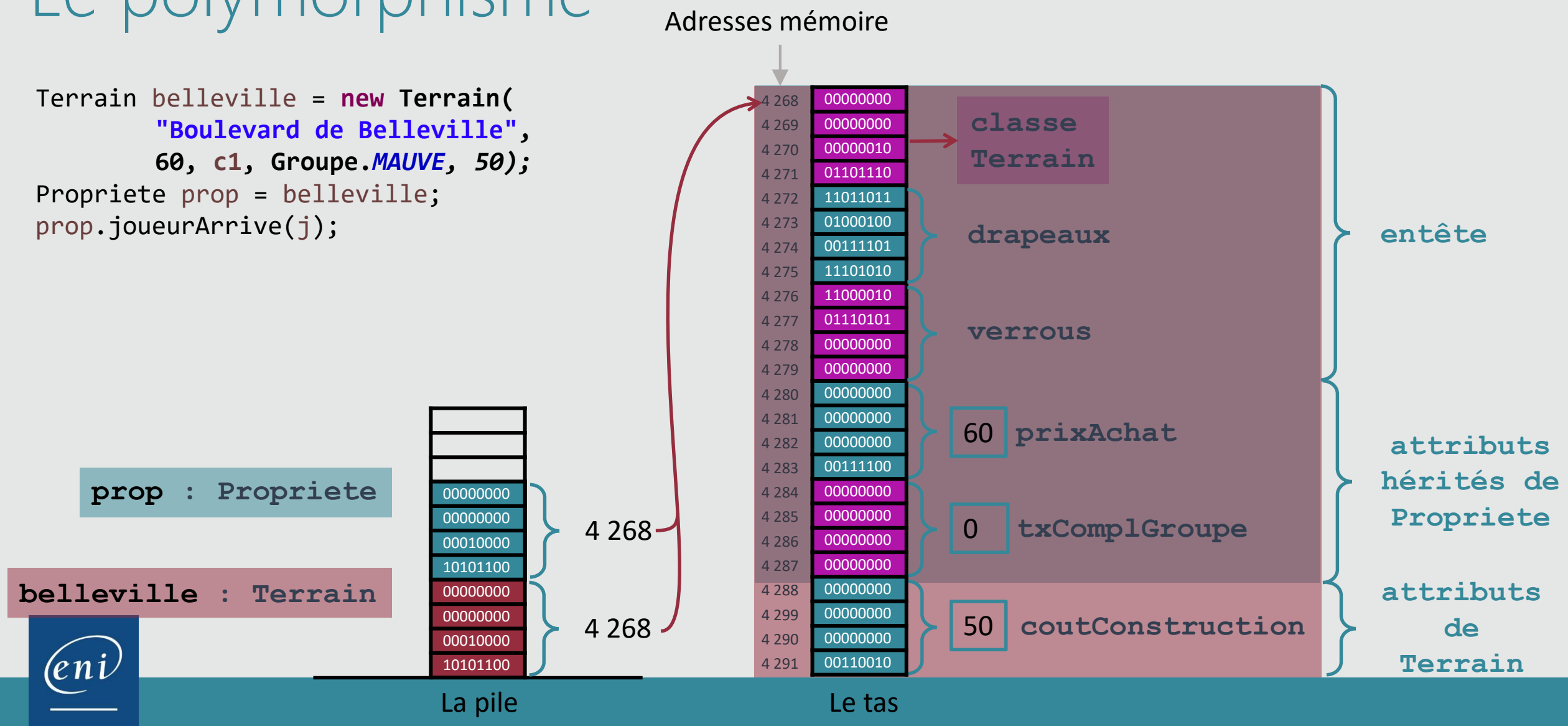
Le polymorphisme

- Le polymorphisme est la capacité de choisir dynamiquement la méthode qui correspond au type réel de l'objet.



Le polymorphisme

```
Terrain belleville = new Terrain(
        "Boulevard de Belleville",
        60, c1, Groupe.MAUVE, 50);
Propriete prop = belleville;
prop.joueurArrive(j);
```



TP