Java
Héritage

Objectifs

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

L’héritage

- Préambule

L’héritage

- Préambule

Des « cartes »

Une multitude de comportements

L’héritage

Carte
-couleur : String -valeur : int
+Carte(couleur : String, valeur : int) +estAutorise(cartePrecedente : Carte) : boolean

CartePasseTonTour
-couleur : String
+CartePasseTonTour (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

CarteReverse
-couleur : String
+CarteReverse (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

CartePiocheDeux
-couleur : String
+CartePiocheDeux (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

CarteCouleur
-couleur : String
+CarteCouleur (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

CartePiocheQuatre
-couleur : String
+CartePiocheQuatre (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

L’héritage

Carte
-couleur : String -valeur : int
+Carte(couleur : String, valeur : int) +estAutorise(cartePrecedente : Carte) : boolean

CartePasseTonTour
-couleur : String
+CartePasseTonTour (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

CarteReverse
-couleur : String
+CarteReverse (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

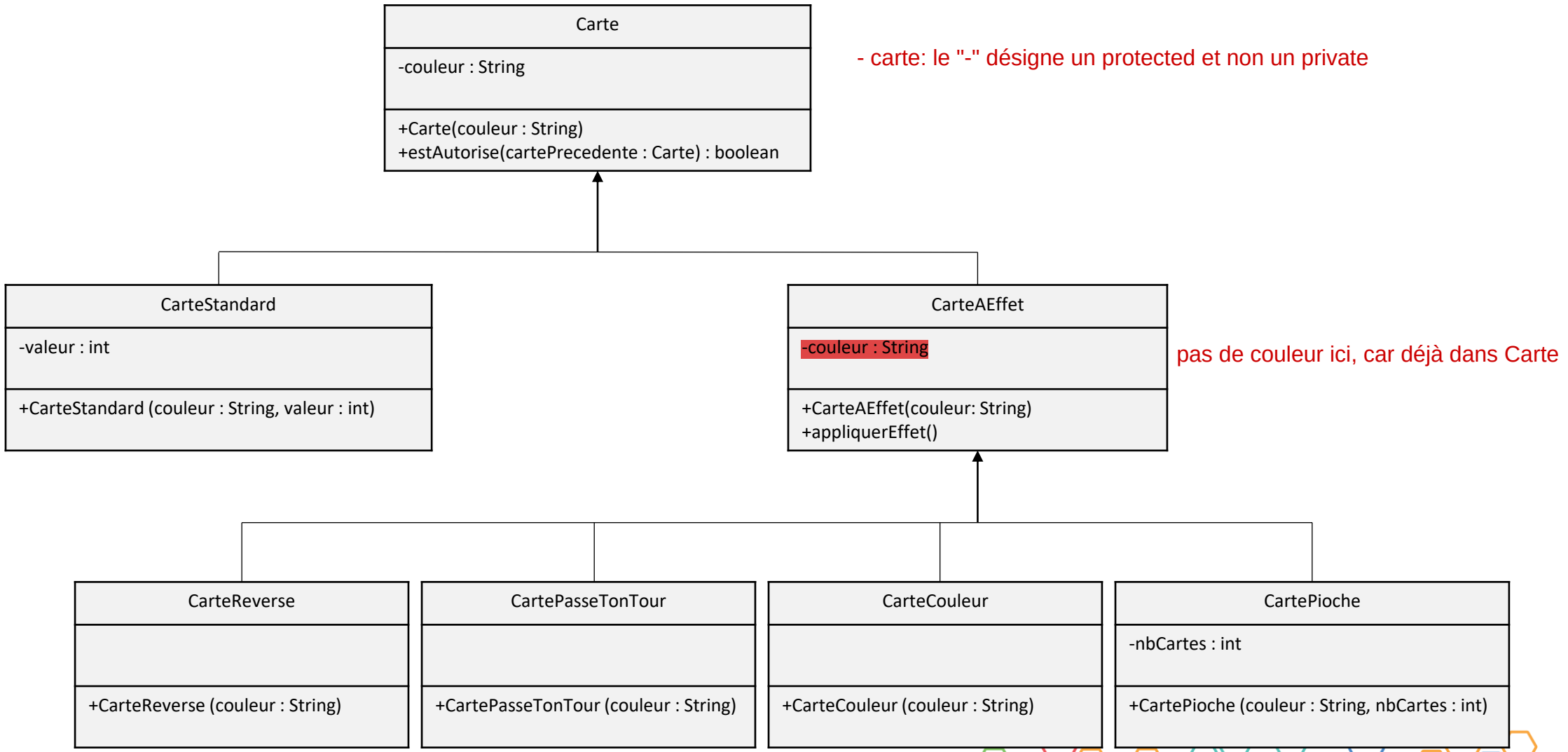
CartePiocheDeux
-couleur : String
+CartePiocheDeux (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

CarteCouleur
-couleur : String
+CarteCouleur (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

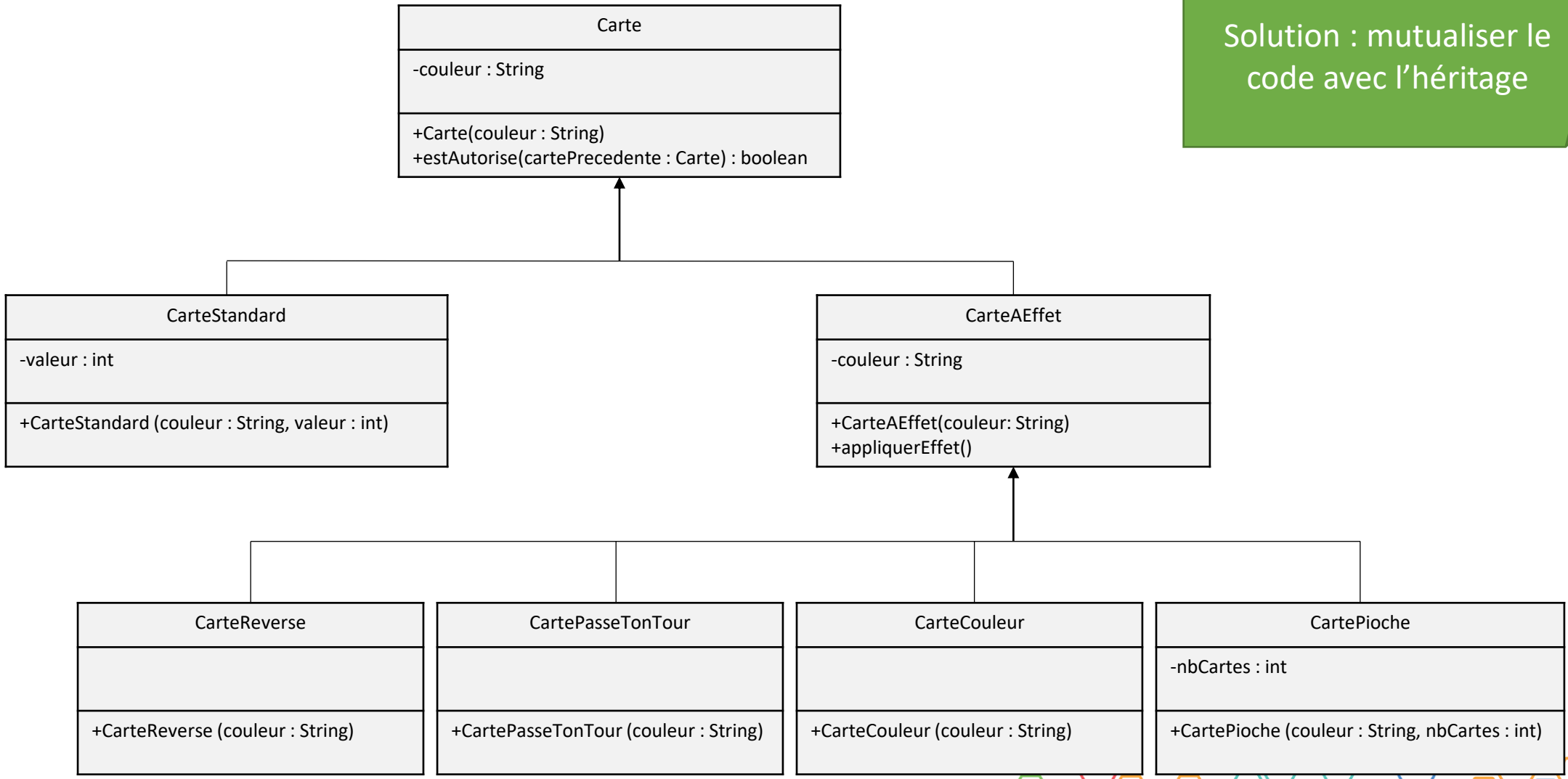
CartePiocheQuatre
-couleur : String
+CartePiocheQuatre (couleur : String) +estAutorise(cartePrecedente : Carte) : boolean +appliquerEffet()

Inconvénient : du code dupliqué à tous les étages

L’héritage


Carte
-couleur : String
+Carte(couleur : String)
+estAutorise(cartePrecedente : Carte) : boolean
- carte: le "-" désigne un protected et non un private
CarteStandard
-valeur : int
+CarteStandard (couleur : String, valeur : int)
CarteAEffet
-couleur : String
+CarteAEffet(couleur: String)
+appliquerEffet()
pas de couleur ici, car déjà dans Carte
CarteReverse
+CarteReverse (couleur : String)
CartePasseTonTour
+CartePasseTonTour (couleur : String)
CarteCouleur
+CarteCouleur (couleur : String)
CartePioche
-nbCartes : int
+CartePioche (couleur : String, nbCartes : int)

L’héritage



Solution : mutualiser le code avec l’héritage

L'héritage

protected = private sauf que les classes filles peuvent y accéder

```
public class Carte {
    protected String couleur;

    public Carte(String couleur) {
        this.couleur = couleur;
    }

    public void jouerCarte() {
        System.out.println("Le joueur pose une carte " + couleur);
    }

    public boolean estAutorise(Carte cartePrecedente) {
        return cartePrecedente.getCouleur().equals(this.couleur);
    }
```

La classe parent

L'héritage

```
public class Carte {
    protected String couleur;

    public Carte(String couleur) {
        this.couleur = couleur;
    }

    public void jouerCarte() {
        System.out.println("Le joueur pose une carte " + couleur);
    }

    public boolean estAutorise(Carte cartePrecedente) {
        return cartePrecedente.getCouleur().equals(this.couleur);
    }
```

La classe parent

L'héritage

extends: hérite

La classe enfant

```
public class CarteStandard extends Carte {
    private int valeur;

    public CarteStandard(String couleur, int valeur) {
        super(couleur);
        this.valeur = valeur;
    }

    @Override
    public void jouerCarte() {
        System.out.println(
                "Le joueur pose un " + valeur + " " + couleur);
    }
```

L'héritage

```
public class CarteStandard extends Carte {
    private int valeur;

    public CarteStandard(String couleur, int valeur) {
        super(couleur);
        this.valeur = valeur;
    }

    @Override
    public void jouerCarte() {
        System.out.println(
                "Le joueur pose un " + valeur + " " + couleur);
    }
```

extends
est le mot clé pour
indiquer un héritage

On a écrasé/remplacé la méthode de la classe mère

L'héritage

```
public class CarteStandard extends Carte {
    private int valeur;

    public CarteStandard(String couleur, int valeur) {
        super(couleur);
        this.valeur = valeur;
    }

    @Override
    public void jouerCarte() {
        System.out.println(
                "Le joueur pose un " + valeur + " " + couleur);
    }
```

définition d'un attribut supplémentaire **en plus** de celui défini dans Carte

L'héritage

```
public class CarteStandard extends Carte {
    private int valeur;

    public CarteStandard(String couleur, int valeur) {
        super(couleur);
        this.valeur = valeur;
    }

    @Override
    public void jouerCarte() {
        System.out.println(
                "Le joueur pose un " + valeur + " " + couleur);
    }
```

super() fait appel au constructeur de la classe parent

Dans le constructeur:
super <=> new carte Carte
Eclipse crée un constructeur en auto

L'héritage

```
public class CarteStandard extends Carte {
    private int valeur;

    public CarteStandard(String couleur, int valeur) {
                                    valeur;
    }

    @Override
    public void jouerCarte() {
        System.out.println(
                "Le joueur pose un " + valeur + " " + couleur);
    }
```

L'annotation "@Override" indique une substitution de méthode

L'héritage

```
public class CarteStandard extends Carte {
    private int valeur;

    public CarteStandard(String couleur, int valeur) {
                                    valeur;
    }

    @Override
    public void jouerCarte() {
        System.out.println(
                "Le joueur pose un " + valeur + " " + couleur);
    }
```

L'annotation "@Override" indique une substitution de méthode

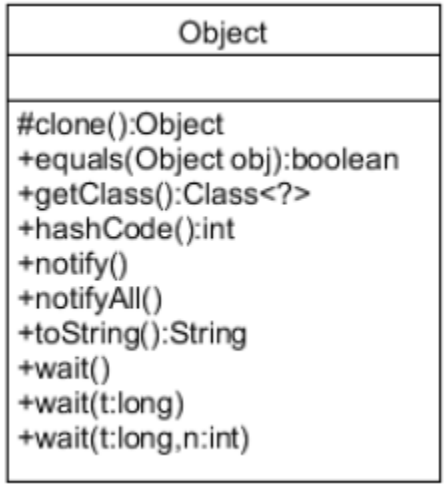
La méthode "jouerCarte" étant substituée, la version définie dans la classe Carte est remplacée par celle-ci

L’héritage

La classe Object

Classe Ultime

- Toute classe sans héritage explicite hérite implicitement de la classe Object
  - Donc, directement ou indirectement, toute classe hérite de Object. « Tout est objet »
  - Donc dans toutes les classes, il est possible d’appeler les méthodes publiques définies dans Object
  - Donc dans toutes les classes, il est possible de substituer les méthodes publiques et protégées de Object



La classe object n'a pas d'attribut

equals permet de comparer des objets (vérifie l'adresse mémoire)

toString() => affiche l'adresse mémoire

L'héritage

Substitution de la méthode toString()

```java
public class Carte {
    protected String couleur;

    public Carte(String couleur) {
        this.couleur = couleur;
    }

    @Override
    public String toString() {
        return "couleur : " + this.couleur;
    }
```

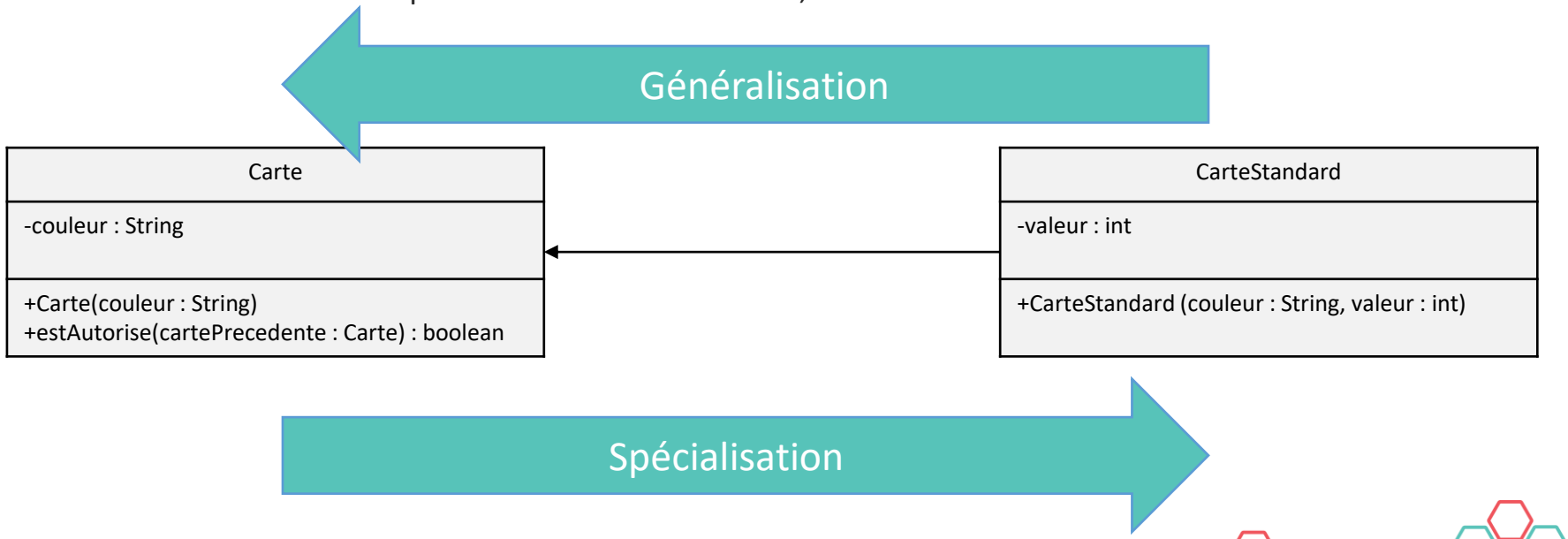
Cette version est toujours mieux que le system.out.println

```java
public class CarteStandard extends Carte {
    private int valeur;

    public CarteStandard(String couleur, int valeur) {
        super(couleur);
        this.valeur = valeur;
    }

    @Override
    public String toString() {
        return "valeur : "
                + this.valeur
                + ", couleur : "
                + this.couleur;
    }
```

L'héritage

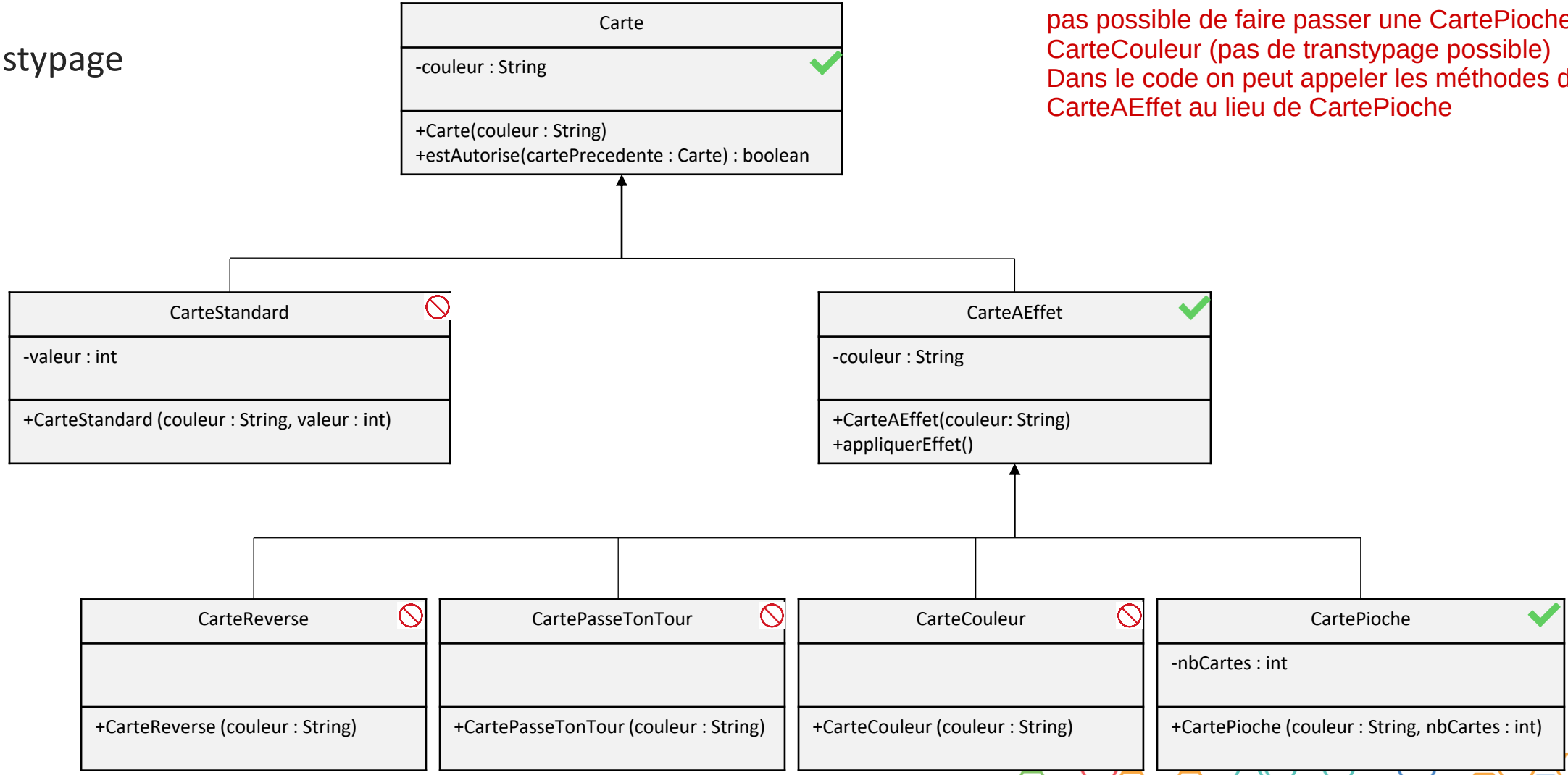
Généralisation et spécialisation

- Comment savoir si un héritage est possible (et pertinent) entre deux classes ?
  - Utilisateur de « est un cas particulier de ». Par exemple
    - « De est un cas particulier de Carte » → Faux, De n'hérite donc pas de Carte
    - « CarteStandard est un cas particulier de Carte » → Vrai, CarteStandard hérite de Carte



L'héritage

Transtypage

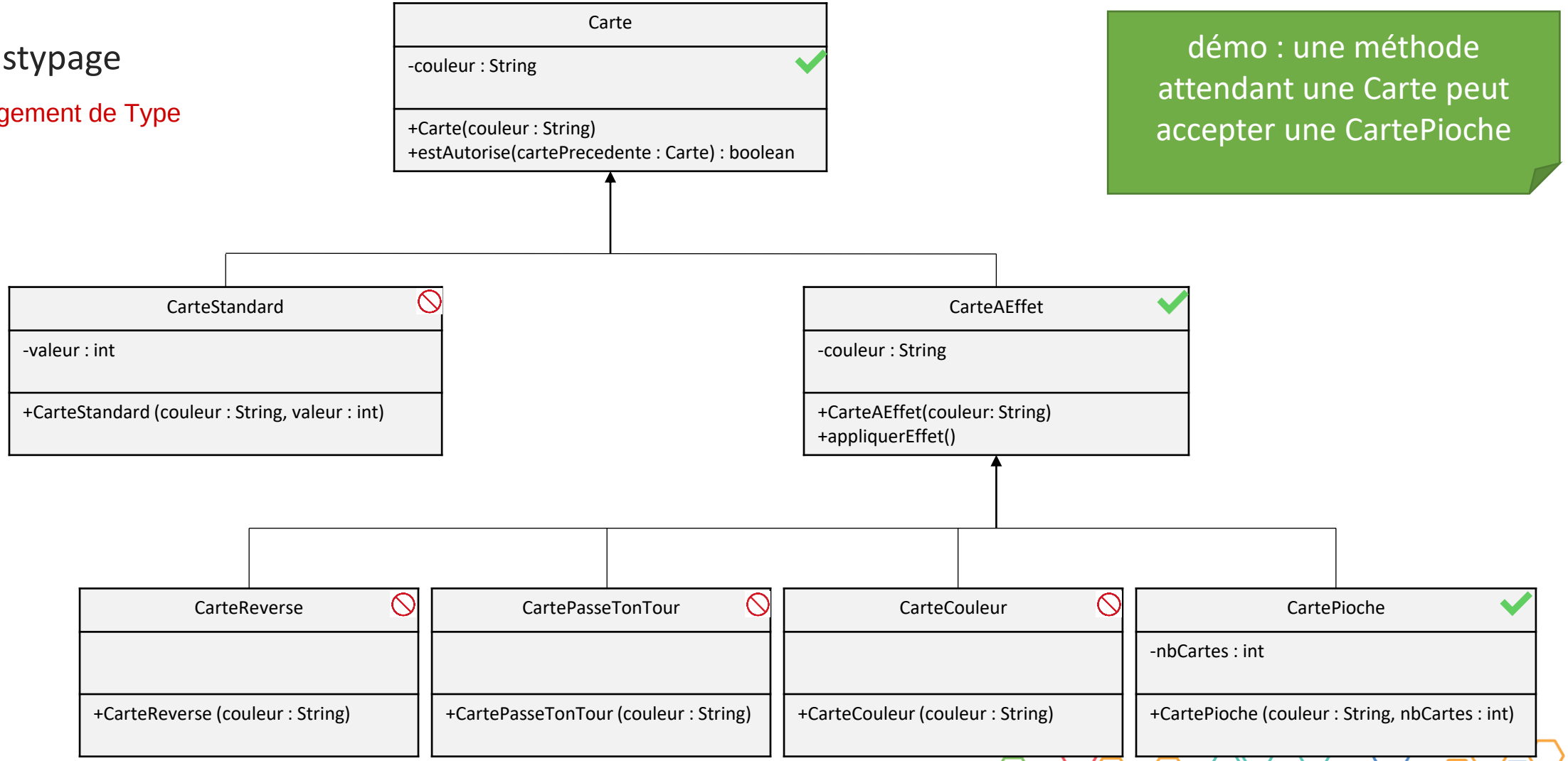


pas possible de faire passer une CartePioche en CarteCouleur (pas de transtypage possible)
Dans le code on peut appeler les méthodes de CarteAEffet au lieu de CartePioche

L'héritage

Transtypage

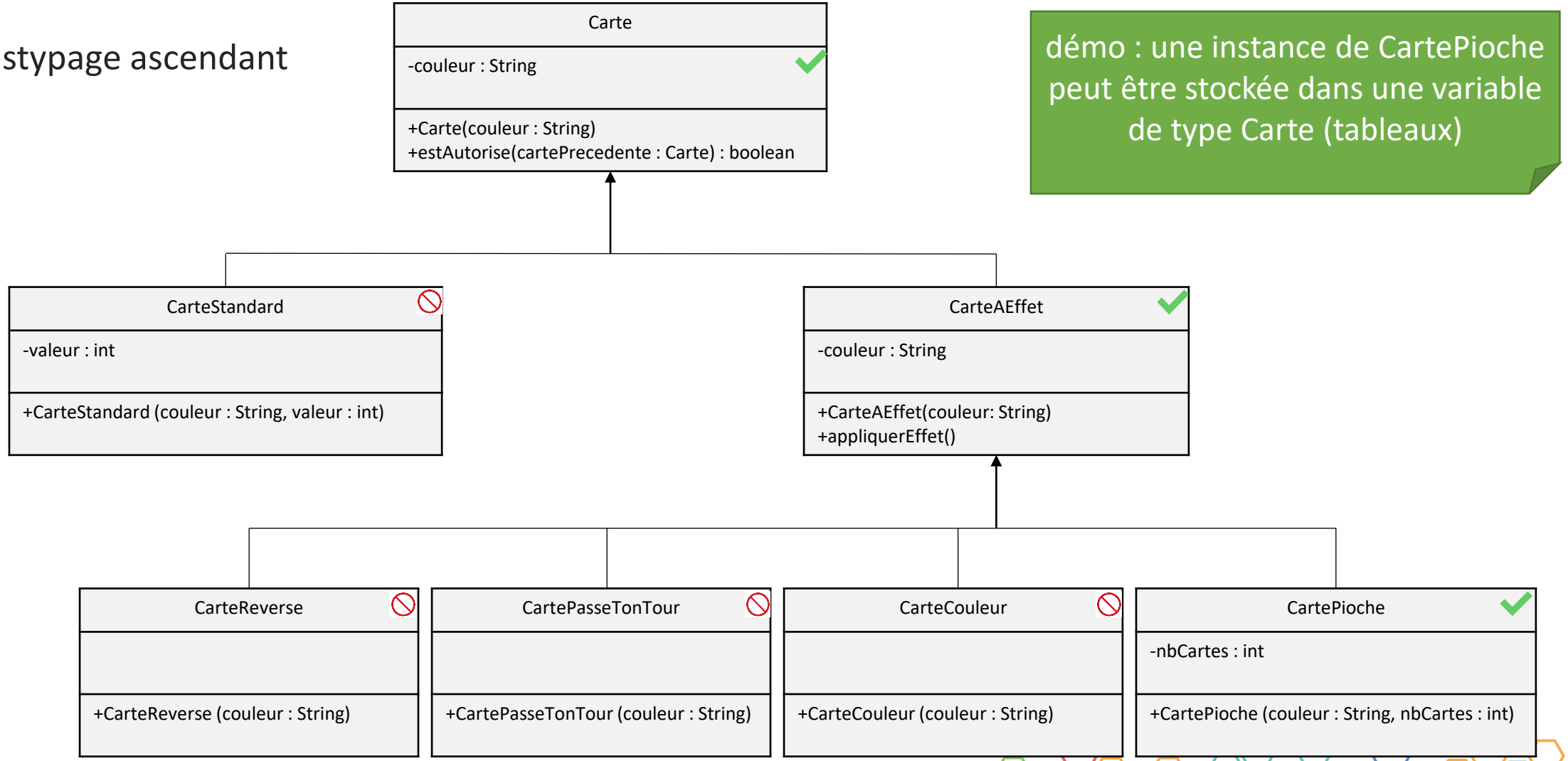
Changement de Type



démo : une méthode attendant une Carte peut accepter une CartePioche

L'héritage

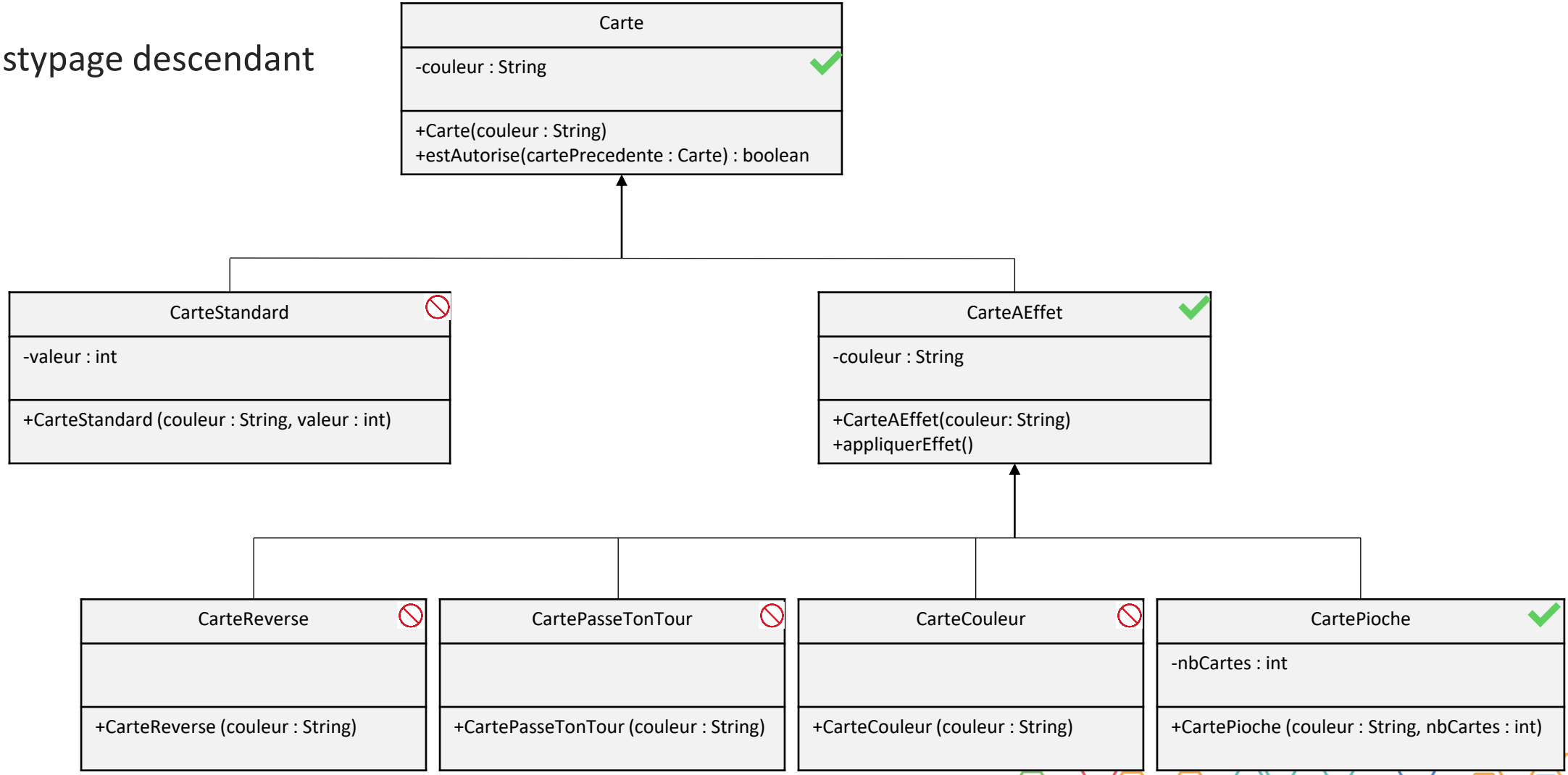
Transtypage ascendant



démo : une instance de CartePioche peut être stockée dans une variable de type Carte (tableaux)

L'héritage

Transtypage descendant



L’héritage

Transtypage descendant

```
public static void main(String[] args) {
    Carte carte = new CarteStandard("bleue", 5);

    // Affichage impossible : la classe "Carte"
    // ne porte par l'information de la valeur
    //System.out.println(carte.getValeur());

    // On cast donc notre carte en "CarteStandard"
    // afin d'accéder à cette information
    if (carte instanceof CarteStandard) {
        CarteStandard cs = (CarteStandard) carte;
        System.out.println(cs.getValeur());
    }
}
```

Autorisé car Carte classe mère de CarteStandard

L'héritage

Transtypage descendant

```
public static void main(String[] args) {
    Carte carte = new CarteStandard("bleue", 5);

    // Affichage impossible : la classe "Carte"
    // ne porte par l'information de la valeur
    //System.out.println(carte.getValeur());

    // On cast donc notre carte en "CarteStandard"
    // afin d'accéder à cette information
    if (carte instanceof CarteStandard) {
        CarteStandard cs = (CarteStandard) carte;
        System.out.println(cs.getValeur());
    }
}
```

instanceof est le mot clé permettant de tester le type de l'instance

on a changé le type de Carte en CarteStandard par: (CarteStandard) carte

(CarteStandard) permet d'effectuer un « cast » de l'instance carte en instance de type CarteStandard

L’héritage

Le polymorphisme

- Le polymorphisme est la capacité de Java à choisir dynamiquement la méthode qui correspond au type réel de l’objet

L’héritage

Le polymorphisme

- Le polymorphisme est la capacité de Java à choisir dynamiquement la méthode qui correspond au type réel de l’objet

démo : la méthode « crier » des animaux

Atelier 5
Encore chez le médecin

Java
Héritage