Java : classes et interfaces

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

Java

Qu'est ce qu'une classe en POO ?

- Ça correspond à un plan, un moule, une usine...
- C'est une description abstraite d'un type d'objets
- Elle représente un ensemble d'objets ayant les mêmes propriétés statiques (attributs) et dynamiques (méthodes)

Qu'est ce que c'est la notion d'instance ?

- Une instance correspond à un objet créé à partir d'une classe (via le constructeur)

Java

Qu'est ce qu'une classe en POO ?

- Ça correspond à un plan, un moule, une usine...
- C'est une description abstraite d'un type d'objets
- Elle représente un ensemble d'objets ayant les mêmes propriétés statiques (attributs) et dynamiques (méthodes)

Qu'est ce que c'est la notion d'instance ?

- Une instance correspond à un objet créé à partir d'une classe (via le constructeur)
- L'instanciation : création d'un objet d'une classe

Java

Qu'est ce qu'une classe en POO ?

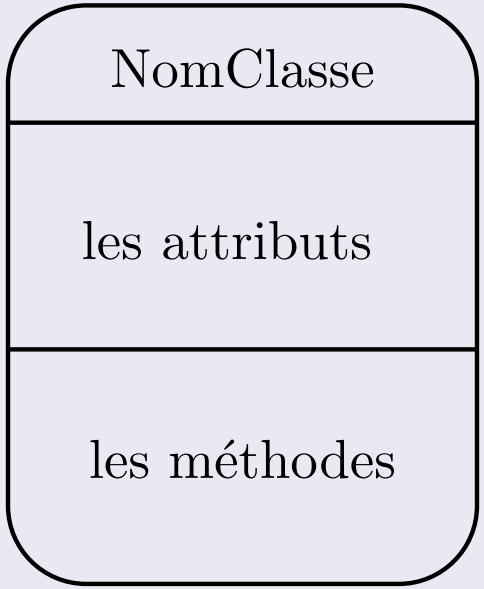
- Ça correspond à un plan, un moule, une usine...
- C'est une description abstraite d'un type d'objets
- Elle représente un ensemble d'objets ayant les mêmes propriétés statiques (attributs) et dynamiques (méthodes)

Qu'est ce que c'est la notion d'instance ?

- Une instance correspond à un objet créé à partir d'une classe (via le constructeur)
- L'instanciation : création d'un objet d'une classe
- instance $\equiv$ objet

Java

De quoi est composé une classe ?



- Attribut : [visibilité] + type + nom
- Méthode : [visibilité] + valeur de retour + nom + arguments ≡ signature : exactement comme les fonctions en procédurale

Java

Particularité du **Java**

- Toutes les classes héritent implicitement (pas besoin d'ajouter `extends`) d'une classe mère `Object`.
- La classe `Object` contient plusieurs méthodes telles que `toString()` (pour transformer un objet en chaîne de caractère), `clone()` (pour cloner)...
- Le mot-clé `this` permet de désigner l'objet courant.
- Contrairement à certains LOO, le `this` n'est pas obligatoire si aucune ambiguité ne se présente.

Java

Commençons par créer un nouveau projet Java

- Aller dans `File > New > Java Project`
- Remplir le champ `Project name:` avec `SecondJavaProject` puis cliquer sur `Next`
- Valider en cliquant sur `Finish`

Java

Commençons par créer un nouveau projet Java

- Aller dans `File > New > Java Project`
- Remplir le champ `Project name:` avec `SecondJavaProject` puis cliquer sur `Next`
- Valider en cliquant sur `Finish`

Créons deux packages `org.eclipse.test` et `org.eclipse.model`

- Aller dans `File > New > Package`
- Saisir le nom du premier package et valider
- Refaire la même chose pour le second

Java

Créons deux classes

- Une classe `Personne` dans `org.eclipse.model` contenant trois attributs : `num`, `nom` et `prénom`
- Une classe `Main` dans `org.eclipse.test` contenant le `public static void main` dans lequel on instanciera la classe `Personne`

Java

Créons deux classes

- Une classe `Personne` dans `org.eclipse.model` contenant trois attributs : `num`, `nom` et `prénom`
- Une classe `Main` dans `org.eclipse.test` contenant le `public static void main` dans lequel on instanciera la classe `Personne`

Créons les classes

- Aller dans `File > New > Class`
- Saisir le nom du package et celui de la classe
- Pour la classe `Main`, cocher la case `public static void main (String[] args)`

Java

Contenu de la classe `Personne`

```
package org.eclipse.model;

public class Personne {
    int num;
    String nom;
    String prenom;
}
```

Remarques

- En Java, toute classe a un constructeur par défaut sans paramètres.
- Par défaut, la visibilité des attributs, en **Java**, est `package`.
- Donc, les attributs ne seront pas accessibles depuis la classe `Main` qui se situe dans un package différent (`org.eclipse.test`)
- Donc, changeons la visibilité des trois attributs de la classe `Personne`

Java

Nouveau contenu de la classe `Personne`

```
package org.eclipse.model;

public class Personne {
    public int num;
    public String nom;
    public String prenom;
}
```

Java

Contenu de la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
    // TODO Auto-generated method stub

    }
}
```

Java

Hypothèse

- Si on voudrait créer un objet de la classe `Personne` avec les valeurs `1`, `wick` et `john`

Java

Hypothèse

- Si on voudrait créer un objet de la classe `Personne` avec les valeurs `1`, `wick` et `john`

Étape 1 : déclaration d'un objet (objet non créé)

```
Personne personne;
```

Java

Hypothèse

- Si on voudrait créer un objet de la classe `Personne` avec les valeurs `1`, `wick` et `john`

Étape 1 : déclaration d'un objet (objet non créé)

```
Personne personne;
```

Étape 2 : instanciation de la classe `Personne` (objet créé)

```
personne = new Personne();
```

Java

Hypothèse

- Si on voudrait créer un objet de la classe `Personne` avec les valeurs `1`, `wick` et `john`

Étape 1 : déclaration d'un objet (objet non créé)

```
Personne personne;
```

Étape 2 : instanciation de la classe `Personne` (objet créé)

```
personne = new Personne();
```

On peut faire déclaration + instanciation

```
Personne personne = new Personne();
```

Java

Affectons les valeurs aux différents attributs

```
personne.num = 1;
personne.nom = "wick";
personne.prenom = "john";
```

Java

Affectons les valeurs aux différents attributs

```
personne.num = 1;
personne.nom = "wick";
personne.prenom = "john";
```

Pour être sûr que les valeurs ont bien été affectées aux attributs, on affiche

```
System.out.println(personne)
```

Java

Contenu de la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.num = 1;
        personne.nom = "wick";
        personne.prenom = "john";
        System.out.println(personne);
    }
}
```

Java

Contenu de la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.num = 1;
        personne.nom = "wick";
        personne.prenom = "john";
        System.out.println(personne);
    }
}
```

En exécutant, le résultat est :

```
org.eclipse.model.Personne@7852e922
```

Java

Explication

- Pour afficher les détails d'un objet, il faut que la méthode `toString()` soit implémentée
- Pour la générer, clic droit sur la classe `Personne`, aller dans `source > Generate toString()...`, sélectionner les champs à afficher puis valider.

Java

Explication

- Pour afficher les détails d'un objet, il faut que la méthode `toString()` soit implémentée
- Pour la générer, clic droit sur la classe `Personne`, aller dans `source > Generate toString()...`, sélectionner les champs à afficher puis valider.

Le code de la méthode `toString()`

```
@Override
public String toString() {
    return "Personne [num=" + num + ", nom=" + nom +
        ", prenom=" + prenom + "]";
}
```

Java

`@Override`

- Une annotation (appelé aussi décorateur par **MicroSoft**)
- Pour nous rappeler qu'on redéfinit une méthode qui appartient à la classe mère (ici `Object`)

Java

`@Override`

- Une annotation (appelé aussi décorateur par **MicroSoft**)
- Pour nous rappeler qu'on redéfinit une méthode qui appartient à la classe mère (ici `Object`)

En exécutant, le résultat est :

```
Personne [num=1, nom=wick, prenom=john]
```

Java

Hypothèse

Supposant que l'on n'accepte pas de valeur négative pour l'attribut `num` de la classe `Personne`

Java

Hypothèse

Supposant que l'on n'accepte pas de valeur négative pour l'attribut `num` de la classe `Personne`

Démarche

1. Bloquer l'accès directe aux attributs (mettre la visibilité à `private`)
2. Définir des méthodes publiques qui contrôlent l'affectation de valeurs aux attributs (les `setter`)

Java

Hypothèse

Supposant que l'on n'accepte pas de valeur négative pour l'attribut `num` de la classe `Personne`

Démarche

1. Bloquer l'accès directe aux attributs (mettre la visibilité à `private`)
2. Définir des méthodes publiques qui contrôlent l'affectation de valeurs aux attributs (les `setter`)

Convention

- Mettre la visibilité `private` ou `protected` pour tous les attributs
- Mettre la visibilité `public` pour toutes les méthodes

Java

Mettons la visibilité `private` pour tous les attributs de la classe `Personne`

```
package org.eclipse.model;

public class Personne {
    private int num;
    private String nom;
    private String prenom;

    @Override
    public String toString() {
        return "Personne [num=" + num + ", nom=" + nom + ",
           prenom=" +     prenom + "]";
    }
}
```

Java

Mettons la visibilité `private` pour tous les attributs de la classe `Personne`

```
package org.eclipse.model;

public class Personne {
    private int num;
    private String nom;
    private String prenom;

    @Override
    public String toString() {
        return "Personne [num=" + num + ", nom=" + nom + ",
           prenom=" +     prenom + "]";
    }
}
```

Dans la classe `Main`, les trois lignes suivantes sont soulignées en rouge

```
personne.num = 1;
personne.nom = "wick";
personne.prenom = "john";
```

Java

Explication

- Les attributs sont privés, donc aucun accès direct ne sera autorisé

Java

Explication

- Les attributs sont privés, donc aucun accès direct ne sera autorisé

Solution : générer les setters

- Faire clic droit sur la classe `Personne`
- Aller dans `Source > Generate Getters and Setters...`
- Cliquer sur chaque attribut et cocher la case `setNomAttribut(...)`
- Valider

Java

Les trois méthodes générées

```
public void setNum(int num) {
    this.num = num;
}
public void setNom(String nom) {
    this.nom = nom;
}
public void setPrenom(String prenom) {
    this.prenom = prenom;
}
```

Java

Les trois méthodes générées

```
public void setNum(int num) {
    this.num = num;
}
public void setNom(String nom) {
    this.nom = nom;
}
public void setPrenom(String prenom) {
    this.prenom = prenom;
}
```

Modifions `setNum()` pour ne plus accepter de valeurs inférieures à 1

```
public void setNum(int num) {
    if (num >= 1){
        this.num = num;
    }
}
public void setNom(String nom) {
    this.nom = nom;
}
public void setPrenom(String prenom) {
    this.prenom = prenom;
}
```

Java

Mettons à jour la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.setNum(1);
        personne.setNom("wick");
        personne.setPrenom("john");
        System.out.println(personne);
    }
}
```

Java

Mettons à jour la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.setNum(1);
        personne.setNom("wick");
        personne.setPrenom("john");
        System.out.println(personne);
    }
}
```

En exécutant, le résultat est :

```
Personne [num=1, nom=wick, prenom=john]
```

Java

Mettons à jour la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.setNum(-1);
        personne.setNom("wick");
        personne.setPrenom("john");
        System.out.println(personne);
    }
}
```

Java

Mettons à jour la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.setNum(-1);
        personne.setNom("wick");
        personne.setPrenom("john");
        System.out.println(personne);
    }
}
```

En exécutant, le résultat est :

```
Personne [num=0, nom=wick, prenom=john]
```

Java

Hypothèse

Si on voudrait afficher les attributs (privés) de la classe `Personne`, un par un, dans la classe `Main` sans passer par le `toString()`

Java

Hypothèse

Si on voudrait afficher les attributs (privés) de la classe `Personne`, un par un, dans la classe `Main` sans passer par le `toString()`

Démarche

Définir des méthodes qui retournent les valeurs des attributs (les `getter`)

Java

Pour générer les getters

- Faire clic droit sur la classe `Personne`
- Aller dans `Source > Generate Getters and Setters...`
- Cliquer sur chaque attribut et cocher la case `getNomAttribut()`
- Valider

Java

Mettons à jour la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.setNum(1);
        personne.setNom("wick");
        personne.setPrenom("john");
        System.out.println(personne.getNum() + " " + personne.
           getNom() + " " + personne.getPrenom());
    }
}
```

Java

Mettons à jour la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.setNum(1);
        personne.setNom("wick");
        personne.setPrenom("john");
        System.out.println(personne.getNum() + " " + personne.
           getNom() + " " + personne.getPrenom());
    }
}
```

En exécutant, le résultat est :

```
1 wick john
```

Java

Remarques

- Par défaut, toute classe en Java un constructeur par défaut sans paramètre
- Pour simplifier la création d'objets, on peut définir un nouveau constructeur qui prend en paramètre plusieurs attributs de la classe

Java

Remarques

- Par défaut, toute classe en Java un constructeur par défaut sans paramètre
- Pour simplifier la création d'objets, on peut définir un nouveau constructeur qui prend en paramètre plusieurs attributs de la classe

Pour générer un constructeur avec paramètre

1. Faire clic droit sur la classe `Personne`
2. Aller dans `Source > Generate Constructor using Fields...`
3. Vérifier que toutes les cases sont cochées et valider

Java

Le constructeur généré

```
public Personne(int num, String nom, String prenom) {
    super();
    this.num = num;
    this.nom = nom;
    this.prenom = prenom;
}
```

Java

Le constructeur généré

```
public Personne(int num, String nom, String prenom) {
    super();
    this.num = num;
    this.nom = nom;
    this.prenom = prenom;
}
```

Pour préserver la cohérence, il faut que le constructeur contrôle la valeur de l'attribut `num`

```
public Personne(int num, String nom, String prenom) {
    super();
    if (num >= 1){
        this.num = num;
    }
    this.nom = nom;
    this.prenom = prenom;
}
```

Java

On peut aussi appelé le `setter` dans le constructeur

```
public Personne(int num, String nom, String prenom)
   {
     super();
     this.setNum(num);
     this.nom = nom;
     this.prenom = prenom;
}
```

Java

Dans la classe `Main`, la ligne suivante est soulignée en rouge

```
Personne personne = new Personne();
```

Java

Dans la classe `Main`, la ligne suivante est soulignée en rouge

```
Personne personne = new Personne();
```

Explication

Le constructeur par défaut a été écrasé (il n'existe plus)

Java

Dans la classe `Main`, la ligne suivante est soulignée en rouge

```
Personne personne = new Personne();
```

Explication

Le constructeur par défaut a été écrasé (il n'existe plus)

Solution

Générer un constructeur sans paramètre

Java

Mettons à jour la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.setNum(1);
        personne.setNom("wick");
        personne.setPrenom("john");
        System.out.println(personne.getNum() + " " + personne.getNom()
           + " " + personne.getPrenom());
        Personne personne2 = new Personne(2,"bob","mike");
        System.out.println(personne2);
    }
}
```

Java

Mettons à jour la classe `Main`

```
package org.eclipse.test;

public class Main {

    public static void main(String[] args) {
        Personne personne = new Personne();
        personne.setNum(1);
        personne.setNom("wick");
        personne.setPrenom("john");
        System.out.println(personne.getNum() + " " + personne.getNom()
           + " " + personne.getPrenom());
        Personne personne2 = new Personne(2,"bob","mike");
        System.out.println(personne2);
    }
}
```

En exécutant, le résultat est :

```
1 wick john
Personne [num=2, nom=bob, prenom=mike]
```

Java

Récapitulatif

Les instances d'une même classe ont toutes les mêmes attributs mais pas les mêmes valeurs

Java

Récapitulatif

Les instances d'une même classe ont toutes les mêmes attributs mais pas les mêmes valeurs

Hypothèse

Et si nous voudrions qu'un attribut ait une valeur partagée par toutes les instances (le nombre d'objets instanciés de la classe `Personne`)

Java

Récapitulatif

Les instances d'une même classe ont toutes les mêmes attributs mais pas les mêmes valeurs

Hypothèse

Et si nous voudrions qu'un attribut ait une valeur partagée par toutes les instances (le nombre d'objets instanciés de la classe `Personne`)

Définition

Un attribut dont la valeur est partagée par toutes les instances de la classe est appelée : attribut statique ou attribut de classe

Java

Exemple

- Si on veut créer un attribut contenant le nombre des objets créés à partir de la classe `Personne`
- Notre attribut doit être `static`, sinon chaque objet pourrait avoir sa propre valeur pour cet attribut

Java

Ajoutons un attribut statique `nbrPersonnes` à la liste d'attributs de la classe `Personne`

```
private  static int nbrPersonnes;
```

Java

Ajoutons un attribut statique `nbrPersonnes` à la liste d'attributs de la classe `Personne`

```
private  static int nbrPersonnes;
```

Incrémentons notre compteur de personnes dans les constructeurs

```
public Personne() {
    super();
    nbrPersonnes++;
}
public Personne(int num, String nom, String prenom) {
    super();
    this.setNum(num);
    this.nom = nom;
    this.prenom = prenom;
    nbrPersonnes++;
}
```

Java

Générons un `getter` pour l'attribut static `nbrPersonnes`

```
public static int getNbrPersonnes() {
    return nbrPersonnes;
}
```

Java

Générons un `getter` pour l'attribut static `nbrPersonnes`

```
public static int getNbrPersonnes() {
    return nbrPersonnes;
}
```

Testons cela dans la classe `Main`

```
Personne personne = new Personne();
personne.setNum(1);
personne.setNom("wick");
personne.setPrenom("john");
System.out.println(personne.getNum() + " " +personne.getNom() + " " +
  personne.getPrenom());
System.out.println(Personne.getNbrPersonnes());
Personne personne2 = new Personne(2,"bob","mike");
System.out.println(personne2);
System.out.println(Personne.getNbrPersonnes());
```

Java

Générons un `getter` pour l'attribut static `nbrPersonnes`

```
public static int getNbrPersonnes() {
    return nbrPersonnes;
}
```

Testons cela dans la classe `Main`

```
Personne personne = new Personne();
personne.setNum(1);
personne.setNom("wick");
personne.setPrenom("john");
System.out.println(personne.getNum() + " " +personne.getNom() + " " +
  personne.getPrenom());
System.out.println(Personne.getNbrPersonnes());
Personne personne2 = new Personne(2,"bob","mike");
System.out.println(personne2);
System.out.println(Personne.getNbrPersonnes());
```

Le résultat est

```
1 wick john
1
Personne [num=2, nom=bob, prenom=mike]
2
```

Java

Exercice

- Définir une classe `Adresse` avec trois attributs privés (`rue`, `codePostal` et `ville` de type chaîne de caractère
- Définir un constructeur avec trois paramètres, les getters, les setters et la méthode `toString()`
- Dans la classe `Personne`, ajouter un attribut `adresse` de type `Adresse` et définir un nouveau constructeur à quatre paramètres et le getter et le setter de ce nouvel attribut
- Dans la classe `Main`, créer deux objets, un objet de type `Adresse` et un de type `Personne` et lui attribuer l'adresse créée précédemment
- Afficher tous les attributs de cet objet de la classe `Personne`

Java

L'héritage, quand ?

- Lorsque deux ou plusieurs classes partagent plusieurs attributs (et méthodes)
- Lorsqu'une `Classe1` est **une sorte de** `Classe2`

Java

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire

Java

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et et un niveau

Java

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et et un niveau
- Sémantiquement, enseignant et étudiant sont une sorte de personne

Java

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et et un niveau
- Sémantiquement, enseignant et étudiant sont une sorte de personne
- En plus, les deux partagent plusieurs attributs tels que numéro, nom et prénom

Java

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et et un niveau
- Sémantiquement, enseignant et étudiant sont une sorte de personne
- En plus, les deux partagent plusieurs attributs tels que numéro, nom et prénom
- Donc, on peut mettre en commun les attributs numéro, nom et prénom dans une classe `Personne`

Java

Exemple

- Un enseignant a un numéro, un nom, un prénom et un salaire
- Un étudiant a aussi un numéro, un nom, un prénom et et un niveau
- Sémantiquement, enseignant et étudiant sont une sorte de personne
- En plus, les deux partagent plusieurs attributs tels que numéro, nom et prénom
- Donc, on peut mettre en commun les attributs numéro, nom et prénom dans une classe `Personne`
- Les classes `Étudiant` et `Enseignant` hériteront (`extends`) de la classe `Personne`

Java

Particularité du **Java**

- L'héritage multiple n'est pas permis par le langage Java
- C'est-à-dire, une classe ne peut hériter de plusieurs classes différentes

Java

Créons les classes `Etudiant` et `Enseignant`

- Aller dans `File > New > Class`
- Choisir le package `org.eclipse.model`
- Saisir le nom de la classe
- Dans la section `Superclass`, cliquer sur `Browse` et chercher puis sélectionner `Personne`
- Cliquer sur `Ok` et enfin `Finish`
- Refaire la même chose pour la seconde classe

Java

Contenu de la classe `Enseignant`

```
package org.eclipse.model;

public class Enseignant extends Personne {

}
```

Java

Contenu de la classe `Enseignant`

```
package org.eclipse.model;

public class Enseignant extends Personne {

}
```

Contenu de la classe `Etudiant`

```
package org.eclipse.model;

public class Etudiant extends Personne {

}
```

`extends` est le mot-clé à utiliser pour définir une relation d'héritage entre deux classes

Java

Ensuite

- Créer un attribut `niveau` dans la classe `Etudiant` puis générer les getter et setter
- Créer un attribut `salaire` dans la classe `Enseignant` puis générer les getter et setter

Java

Pour créer un objet de type `Enseignant`

```
Enseignant enseignant = new Enseignant();
enseignant.setNum(3);
enseignant.setNom("green");
enseignant.setPrenom("jonas");
enseignant.setSalaire(1700);
System.out.println(enseignant);
```

Java

Pour créer un objet de type `Enseignant`

```
Enseignant enseignant = new Enseignant();
enseignant.setNum(3);
enseignant.setNom("green");
enseignant.setPrenom("jonas");
enseignant.setSalaire(1700);
System.out.println(enseignant);
```

En exécutant, le résultat est :

```
Personne [num=3, nom=green, prenom=jonas]
```

Java

Pour créer un objet de type `Enseignant`

```
Enseignant enseignant = new Enseignant();
enseignant.setNum(3);
enseignant.setNom("green");
enseignant.setPrenom("jonas");
enseignant.setSalaire(1700);
System.out.println(enseignant);
```

En exécutant, le résultat est :

```
Personne [num=3, nom=green, prenom=jonas]
```

Mais on ne voit pas le salaire, pourquoi ?

car on a pas redéfini la méthode `toString()`, on a utilisé celle de la classe mère

Java

Et si on génère le `toString()` dans la classe `Enseignant`

```
@Override
public String toString() {
    return "Enseignant [salaire=" + salaire + "]";
}
```

Java

Et si on génère le `toString()` dans la classe `Enseignant`

```
@Override
public String toString() {
    return "Enseignant [salaire=" + salaire + "]";
}
```

Et si on voudrait appeler le `toString()` de la classe mère à partir de classe fille (`Enseignant`)

```
@Override
public String toString() {
    return super.toString() +  " Enseignant [salaire
       =" + salaire + "]";
}
```

Le mot-clé `super` permet d'appeler une méthode de la classe mère

Java

Comment générer un constructeur à plusieurs paramètres et utiliser celui de la classe mère

- Faire clic droit sur la classe `Enseignant`
- Aller dans `Source > Generate Constructor using Fields...`
- Dans `Select super constructor to invoke`, sélectionner le constructeur à trois paramètres
- Dans `Select fields to initialize`, vérifier que la case `salaire` est sélectionnée et valider

Java

Comment générer un constructeur à plusieurs paramètres et utiliser celui de la classe mère

- Faire clic droit sur la classe `Enseignant`
- Aller dans `Source > Generate Constructor using Fields...`
- Dans `Select super constructor to invoke`, sélectionner le constructeur à trois paramètres
- Dans `Select fields to initialize`, vérifier que la case `salaire` est sélectionnée et valider

Le constructeur généré

```
public Enseignant(int num, String nom, String prenom, int salaire) {
    super(num, nom, prenom);
    this.salaire = salaire;
}
```

Java

Comment générer un constructeur à plusieurs paramètres et utiliser celui de la classe mère

- Faire clic droit sur la classe `Enseignant`
- Aller dans `Source > Generate Constructor using Fields...`
- Dans `Select super constructor to invoke`, sélectionner le constructeur à trois paramètres
- Dans `Select fields to initialize`, vérifier que la case `salaire` est sélectionnée et valider

Le constructeur généré

```
public Enseignant(int num, String nom, String prenom, int salaire) {
    super(num, nom, prenom);
    this.salaire = salaire;
}
```

Maintenant, on peut créer un enseignant ainsi

```
Enseignant enseignant = new Enseignant(3, "green", "jonas", 1700);
```

Java

À partir de la classe `Enseignant`

- On ne peut avoir accès direct à un attribut de la classe mère
- C'est-à-dire, on ne peut faire `this.num` car les attributs ont une visibilité `private`
- Pour modifier la valeur d'un attribut privé de la classe mère, il faut
  - soit utiliser les getters/setters
  - soit mettre la visibilité des attributs de la classe mère à `protected`

Java

On peut créer un objet de la classe `Personne` ainsi

```
Enseignant enseignant = new Enseignant(3, "green", "
   jonas", 1700);
```

Java

On peut créer un objet de la classe `Personne` ainsi

```
Enseignant enseignant = new Enseignant(3, "green", "
   jonas", 1700);
```

Ou ainsi

```
Personne enseignant = new Enseignant(3, "green", "
   jonas", 1700);
```

Java

On peut créer un objet de la classe `Personne` ainsi

```
Enseignant enseignant = new Enseignant(3, "green", "
   jonas", 1700);
```

Ou ainsi

```
Personne enseignant = new Enseignant(3, "green", "
   jonas", 1700);
```

Ceci est faux

```
Enseignant enseignant = new Personne(3, "green", "
   jonas");
```

Java

Remarque

1. Pour connaître la classe d'un objet, on peut utiliser le mot-clé `instanceof`

Java

Remarque

1. Pour connaître la classe d'un objet, on peut utiliser le mot-clé `instanceof`

Exemple

```
Personne enseignant = new Enseignant(3, "green", "
   jonas", 1700);
System.out.println(enseignant instanceof Enseignant)
   ;
// affiche true
System.out.println(enseignant instanceof Personne);
// affiche true
System.out.println(personne instanceof Enseignant);
// affiche false
```

Java

Exercice

1. Créer deux objets de type `Etudiant` et deux objets de type `Enseignant` et stocker les tous dans un seul tableau.
2. Parcourir le tableau et afficher pour chacun soit le `salaire` soit le `niveau`

Java

Exercice

1. Créer deux objets de type `Etudiant` et deux objets de type `Enseignant` et stocker les tous dans un seul tableau.
2. Parcourir le tableau et afficher pour chacun soit le `salaire` soit le `niveau`

Pour parcourir un tableau, on peut utiliser le raccourci de `for`

```
Personne personnes [] = {personne, personne2,
   enseignant};
for(Personne perso : personnes) {
    System.out.println(perso);
}
```

Java

L'agrégation

- C'est une association non-symétrique
- Elle représente une relation de type ensemble/élément

Java

L'agrégation

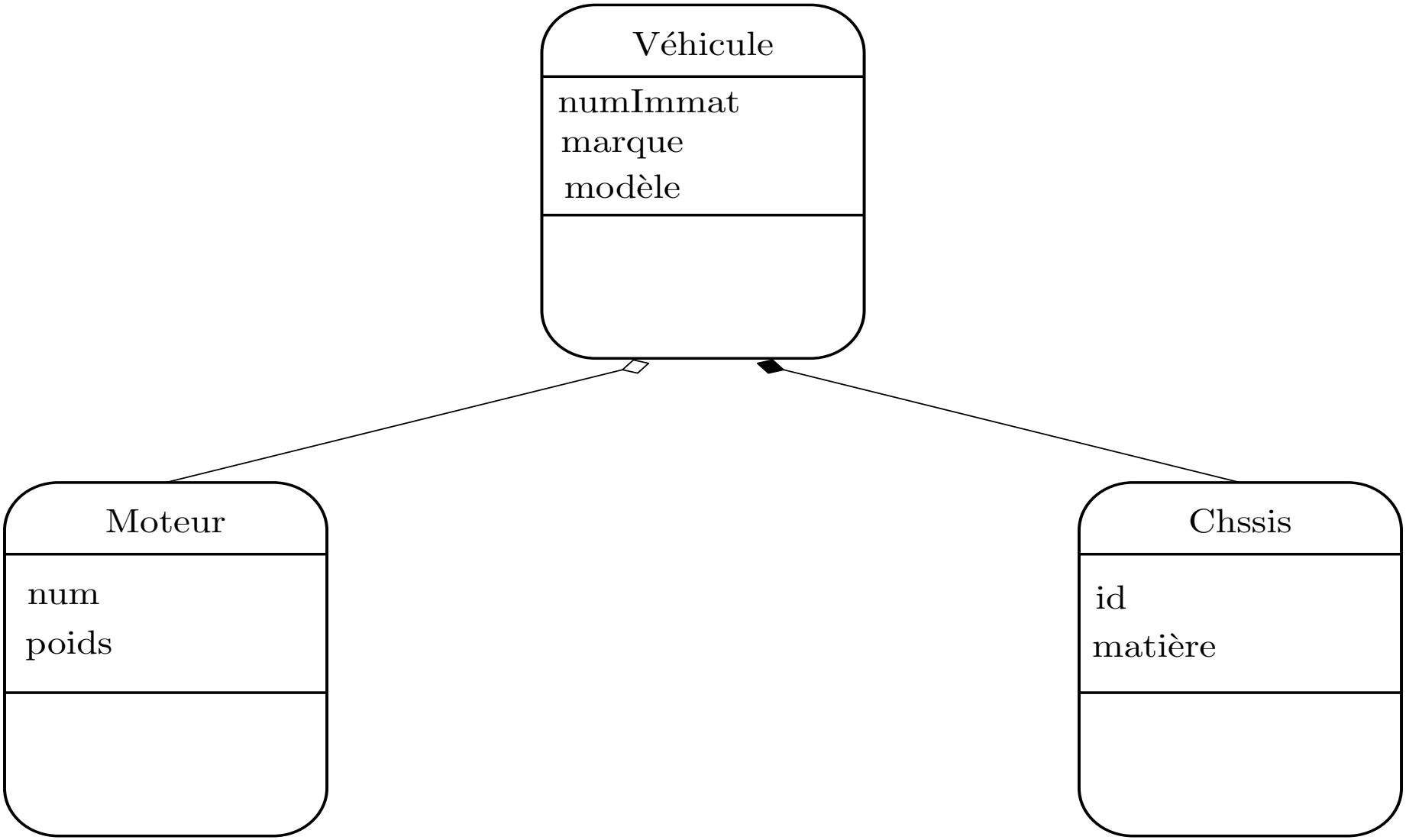
- C'est une association non-symétrique
- Elle représente une relation de type ensemble/élément

La composition

- C'est une agrégation forte
- L'élément n'existe pas sans l'agrégat (l'élément est détruit lorsque l'agrégat n'existe plus)

Java

Comment coder des relations d'agrégation et de composition en Java ?



Java

La classe `Châssis`

```
public class Chassis {
        private int id;
        private String matiere;
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getMatiere() {
                return matiere;
        }
        public void setMatiere(String matiere) {
                this.matiere = matiere;
        }
}
```

Java

La classe `Moteur`

```
public class Moteur {
        private int num;
        private float poids;
        public int getNum() {
                return num;
        }
        public void setNum(int num) {
                this.num = num;
        }
        public float getPoids() {
                return poids;
        }
        public void setPoids(float poids) {
                this.poids = poids;
        }
}
```

La classe `Véhicule`

```
public class Vehivule {
        private Moteur moteur;
        private final Chassis chassis;
        public Vehivule(Chassis chassis) {
                this.chassis = chassis;
        }
        public Moteur getMoteur() {
                return moteur;
        }
        public void setMoteur(Moteur moteur) {
                this.moteur = moteur;
        }
}
```

La classe `Véhicule`

```
public class Vehivule {
        private Moteur moteur;
        private final Chassis chassis;
        public Vehivule(Chassis chassis) {
                this.chassis = chassis;
        }
        public Moteur getMoteur() {
                return moteur;
        }
        public void setMoteur(Moteur moteur) {
                this.moteur = moteur;
        }
}
```

Explication

- un composant est déclaré `final` et instanciable une seule fois au moment de l'instanciation de l'objet composite
- un composant n'a ni getter ni setter
- lorsque le véhiculé est détruit, le châssis le sera aussi

Java

Pour tester

```
public class Main {
  public static void main(String[] args) {
    Chassis chassis = new Chassis();
    chassis.setId(100);
    chassis.setMatiere("fer");
    Vehivule vehivule = new Vehivule(chassis);
    Moteur moteur = new Moteur();
    moteur.setNum(100);
    moteur.setPoids(500f);
    vehivule.setMoteur(moteur);
    // pas de méthode permettant de modifier le châ
       ssis d'un véhicule
  }
}
```

Java

Classe abstraite

- C'est une classe qu'on ne peut instancier
- Les constructeurs peuvent donc être supprimés

Java

Classe abstraite

- C'est une classe qu'on ne peut instancier
- Les constructeurs peuvent donc être supprimés

Si on déclare la classe `Personne` abstraite

```
public abstract class Personne {
    ...
}
```

Java

Classe abstraite

- C'est une classe qu'on ne peut instancier
- Les constructeurs peuvent donc être supprimés

Si on déclare la classe `Personne` abstraite

```
public abstract class Personne {
    ...
}
```

Tout ce code sera souligné en rouge

```
Personne personne = new Personne();
...
Personne personne2 = new Personne(2,"bob","mike");
```

Java

Méthode abstraite

- C'est une méthode non implémentée (sans code)
- Une méthode abstraite doit être déclarée dans une classe abstraite
- Une méthode abstraite doit être implémentée par les classes filles de la classe abstraite

Java

Méthode abstraite

- C'est une méthode non implémentée (sans code)
- Une méthode abstraite doit être déclarée dans une classe abstraite
- Une méthode abstraite doit être implémentée par les classes filles de la classe abstraite

Déclarons une méthode abstraite `afficherNomComplet()` dans `Personne`

```
public abstract void afficherNomComplet();
```

Java

Méthode abstraite

- C'est une méthode non implémentée (sans code)
- Une méthode abstraite doit être déclarée dans une classe abstraite
- Une méthode abstraite doit être implémentée par les classes filles de la classe abstraite

Déclarons une méthode abstraite `afficherNomComplet()` dans `Personne`

```
public abstract void afficherNomComplet();
```

Suite à la déclaration de `afficherNomComplet()` dans `Personne`, les deux classes `Etudiant` et `Enseignant` sont signalées en rouge

Java

Pour implémenter la méthode abstraite

- Placer le curseur sur le nom de la classe
- Dans le menu afficher, choisir `Add unimplemented methods`

Java

Pour implémenter la méthode abstraite

- Placer le curseur sur le nom de la classe
- Dans le menu afficher, choisir `Add unimplemented methods`

Le code généré

```
@Override
public void afficherNomComplet() {
    // TODO Auto-generated method stub

}
```

Java

Pour implémenter la méthode abstraite

- Placer le curseur sur le nom de la classe
- Dans le menu afficher, choisir `Add unimplemented methods`

Le code généré

```
@Override
public void afficherNomComplet() {
    // TODO Auto-generated method stub

}
```

Modifions le code précédent

```
@Override
public void afficherNomComplet() {
    System.out.println(this.getPrenom() + " " + this.getNom());
}
```

Java

Pour tester

```
Enseignant enseignant =  new Enseignant(3,"green","
   jonas",1700);
enseignant.afficherNomComplet();
```

Java

Pour tester

```
Enseignant enseignant =  new Enseignant(3,"green","
   jonas",1700);
enseignant.afficherNomComplet();
```

En exécutant, le résultat est :

```
jonas green
```

Java

Classe finale

C’est une classe qui ne peut avoir de classes filles

Java

Classe finale

C'est une classe qui ne peut avoir de classes filles

Pour tester

Commençons par mettre en commentaire `afficherNomComplet()` dans les trois classes `Personne`, `Enseignant` et `Etudiant`

Java

Classe finale

C'est une classe qui ne peut avoir de classes filles

Pour tester

Commençons par mettre en commentaire `afficherNomComplet()` dans les trois classes `Personne`, `Enseignant` et `Etudiant`

Déclarons la classe `Personne` finale

```
public final class Personne {
    ...
}
```

Java

Classe finale

C'est une classe qui ne peut avoir de classes filles

Pour tester

Commençons par mettre en commentaire `afficherNomComplet()` dans les trois classes `Personne`, `Enseignant` et `Etudiant`

Déclarons la classe `Personne` finale

```
public final class Personne {
    ...
}
```

Les deux classes filles sont affichées en rouge

```
The type Enseignant cannot subclass the final class Personne
```

Java

Méthode finale

C’est une méthode qu’on ne peut redéfinir

Java

Méthode finale

C'est une méthode qu'on ne peut redéfinir

Pour tester

Commençons par supprimer le mot-clé `final` dans la classe `Personne`

Java

Méthode finale

C'est une méthode qu'on ne peut redéfinir

Pour tester

Commençons par supprimer le mot-clé `final` dans la classe `Personne`

Ajoutons une méthode finale `afficherNomComplet()` **dans** `Personne`

```
public final void afficherNomComplet() {
    System.out.println(this.getPrenom() + " " + this.getNom());

}
```

Java

Méthode finale

C'est une méthode qu'on ne peut redéfinir

Pour tester

Commençons par supprimer le mot-clé `final` dans la classe `Personne`

Ajoutons une méthode finale `afficherNomComplet()` **dans** `Personne`

```
public final void afficherNomComplet() {
    System.out.println(this.getPrenom() + " " + this.getNom());

}
```

Si on essaye de redéfinir cette méthode dans `Enseignant`

```
Cannot override the final method from Personne
```

Java

Remarques

- Une classe abstraite ne doit pas forcément contenir une méthode abstraite
- Une classe finale ne doit pas forcément contenir une méthode finale
- Une méthode finale ne doit pas forcément être dans une classe finale

Java

En Java

- Une classe ne peut hériter que d'une seule classe
- Mais elle peut hériter de plusieurs interfaces

Java

Une interface

- déclarée avec le mot-clé `interface`
- comme une classe abstraite (impossible de l'instancier) dont :
  - toutes les méthodes sont abstraites
  - tous les attributs sont constants
- un protocole, un contrat : toute classe qui hérite d'une interface doit implémenter toutes ses méthodes
- pouvant proposée une implémentation par défaut pour les méthodes en utilisant le mot-clé `default`

Java

Pour créer une interface sous **Eclipse**

- Aller dans `File > New > Interface`
- Saisir `org.eclipse.interfaces` dans `Package name`
- Saisir `IMiseEnForme` dans `Name`
- Valider

Java

Contenu généré de l'interface `IMiseEnForme`

```
package org.eclipse.interfaces;

public interface IMiseEnForme {

}
```

Java

Contenu généré de l'interface `IMiseEnForme`

```
package org.eclipse.interfaces;

public interface IMiseEnForme {

}
```

Définissons la signature de deux méthodes dans l'interface `IMiseEnForme`

```
package org.eclipse.interfaces;

public interface IMiseEnForme {
    public void afficherNomMajuscule();
    public void afficherPrenomMajuscule();
}
```

Java

Pour hériter d'une interface, on utilise le mot-clé `implements`

```
public class Personne implements IMiseEnForme {
    ...
}
```

Java

Pour hériter d'une interface, on utilise le mot-clé `implements`

```
public class Personne implements IMiseEnForme {
    ...
}
```

La classe `Personne` est soulignée en rouge

- Placer le curseur sur la classe `Personne`
- Dans le menu affiché, sélectionner `Add unimplemented methods`

Java

Le code généré

```
@Override
public void afficherNomMajuscule() {
    // TODO Auto-generated method stub

}

@Override
public void afficherPrenomMajuscule() {
    // TODO Auto-generated method stub

}
```

Java

Modifions le code de deux méthodes générées

```
@Override
public void afficherNomMajuscule() {
        System.out.println(nom.toUpperCase());
}

@Override
public void afficherPrenomMajuscule() {
        System.out.println(prenom.toUpperCase());

}
```

Java

Pour tester

```
Enseignant enseignant =  new Enseignant(3,"green","
   jonas",1700);
enseignant.afficherNomMajuscule();
enseignant.afficherPrenomMajuscule();
```

Java

Pour tester

```
Enseignant enseignant =  new Enseignant(3,"green","
   jonas",1700);
enseignant.afficherNomMajuscule();
enseignant.afficherPrenomMajuscule();
```

En exécutant, le résultat est :

```
GREEN
JONAS
```

Java

Définissons un attribut `i` dans l'interface `IMiseEnForme`

```
package org.eclipse.interfaces;

public interface IMiseEnForme {

    int i = 5;

    public void afficherNomMajuscule();
    public void afficherPrenomMajuscule();

}
```

Java

Définissons un attribut `i` dans l'interface `IMiseEnForme`

```
package org.eclipse.interfaces;

public interface IMiseEnForme {

    int i = 5;

    public void afficherNomMajuscule();
    public void afficherPrenomMajuscule();

}
```

Vérifier qu'il est impossible de vérifier la valeur de `i` dans le `main` et dans la classe `Personne`

Java

Il est possible de définir une implémentation par défaut pour une méthode d'interface (depuis Java 8)

```
package org.eclipse.interfaces;

public interface IMiseEnForme {
        int i = 5;

        default public void afficherNomMajuscule() {
                System.out.println("Doe");
        }
        public void afficherPrenomMajuscule();
}
```

Java

Il est possible de définir une implémentation par défaut pour une méthode d'interface (depuis Java 8)

```
package org.eclipse.interfaces;

public interface IMiseEnForme {
        int i = 5;

        default public void afficherNomMajuscule() {
                System.out.println("Doe");
        }
        public void afficherPrenomMajuscule();
}
```

Remarque

Les classes filles ne sont pas dans l'obligation d'implémenter les méthodes d'une interface ayant une implémentation par défaut

Java

Remarques

- Une interface peut hériter de plusieurs autres interfaces (mais pas d'une classe)
- Pour cela, il faut utiliser le mot-clé `extends` et pas `implements` car une interface n'implémente jamais de méthodes.

Java

Remarques

- Une interface peut hériter de plusieurs autres interfaces (mais pas d'une classe)
- Pour cela, il faut utiliser le mot-clé `extends` et pas `implements` car une interface n'implémente jamais de méthodes.

Question : une interface est-elle vraiment une classe abstraite ?

Non, car toute classe abstraite hérite de la classe `Object` mais une interface **non**

Java

Énumération

- est un ensemble de constantes
- introduit depuis Java 5
- peut être déclarée, dans une fichier toute seule, dans une classe ou dans une interface

Java

Énumération

- est un ensemble de constantes
- introduit depuis Java 5
- peut être déclarée, dans une fichier toute seule, dans une classe ou dans une interface

Pour créer une énumération sous **Eclipse**

- Aller dans `File > New > Enum`
- Saisir `org.eclipse.enums` dans `Package name`
- Saisir `Sport` dans `Name`
- Valider

Java

Le code généré

```
package org.eclipse.enums;

public enum Sport {

}
```

Java

Le code généré

```
package org.eclipse.enums;

public enum Sport {

}
```

Ajoutons des constantes à cette énumération `Sport`

```
package org.eclipse.enums;

public enum Sport {
        FOOT,
        RUGBY,
        TENNIS,
        CROSS_FIT,
        BASKET
}
```

Java

Pour utiliser cette énumération dans la classe `Main`

```
Sport sport = Sport.BASKET;
System.out.println(sport); // affiche BASKET
```

Java

Pour utiliser cette énumération dans la classe `Main`

```
Sport sport = Sport.BASKET;
System.out.println(sport); // affiche BASKET
```

N'oublions pas d'importer l'énumération dans la classe `Main`

```
import org.eclipse.enums.Sport;
```

Java

On peut aussi définir une énumération comme une classe avec un ensemble d'attributs et de méthodes

```
public enum Sport {

    FOOT("foot",1),
    RUGBY("rugby",7),
    TENNIS("tennis",3),
    CROSS_FIT("cross_fit",4),
    BASKET("basket",6) ;
    private final String nom;
    private final int code;

    Sport(String nom, int code) {
        this.nom = nom;
        this.code = code;
    }
    public String getNom(){ return this.nom; }
    public int getCode(){ return this.code; }
};
```

Java

Récupérer la valeur de l'attribut `code` de la constante `BASKET`

```
Sport sport = Sport.BASKET;
System.out.println(sport.getCode());
// affiche 6
```

Java

Récupérer la valeur de l'attribut `code` de la constante `BASKET`

```
Sport sport = Sport.BASKET;
System.out.println(sport.getCode());
// affiche 6
```

Récupérer l'indice de la constante `BASKET` dans l'énumération

```
Sport sport = Sport.BASKET;
System.out.println(sport.ordinal());
// affiche l'indice ici 4
```

Java

Récupérer la valeur de l'attribut `code` de la constante `BASKET`

```
Sport sport = Sport.BASKET;
System.out.println(sport.getCode());
// affiche 6
```

Récupérer l'indice de la constante `BASKET` dans l'énumération

```
Sport sport = Sport.BASKET;
System.out.println(sport.ordinal());
// affiche l'indice ici 4
```

Transformer l'énumération en tableau et récupérer l'élément d'indice 2

```
System.out.println(Sport.values()[2]);
// affiche Tennis
```