TD 02

Exercice 1 : La classe `Person`

Lisez attentivement le code de la classe `Person` fournie :

```
class Person {
  String name = "";
  int age = -1;

  Person() {};
  Person(String aName, int value){this.name = aName;this.age = value;}
  Person(Person anotherPerson) {this.name = anotherPerson.name;age = anotherPerson.age;}

  public String toString() {return this.name + "(" + this.age + ")";}

  void changeMe(Person aName) {name = aName.name;age = aName.age;}
  void changeYou(Person aName) {aName.name = name;aName.age = this.age;}
  void changeComplete(Person aName) {aName = this;}

  Person getMe() {return this;}
  Person getLikeMe() {return new Person(this);}

  void changeMyAge(int aValue) {age = aValue;}
  void changeYourAge(int aValue) {aValue = age;}
  void changeAName(String aName) {aName = name;}
  void changeAnotherName(String aName) {name = aName;}
  String changeYourName(String aName) {return name;}
}
```

Un certain nombre de constructeurs et de méthodes sont fournies. Aucune erreur de syntaxe n'est à déplorer, mais ça ne veut pas dire pour autant que ce code est sensé ou bien écrit...Tous les programmes suivants sont initialisés selon les mêmes conditions, à savoir :

```
Person alice = new Person();
Person bruno = new Person("alice", 42);
Person clement = new Person(bruno);
```

Q1. Observez les constructeurs. Sont-ils tous nécessaires et bien écrit ? Qu'affiche le programme suivant ?

```
System.out.println(alice + "\t" + bruno + "\t" + clement);
bruno.age = 404;
bruno.name = "bruno";
System.out.println(alice + "\t" + bruno + "\t" + clement);
```

Q2. Observez les trois méthodes `changeMe(...)`, `changeYou(...)` et `changeComplete(...)`. Sont-elles bien écrites ? Qu'affiche le programme suivant ?

```
System.out.println(alice + "\t" + bruno + "\t" + clement);
bruno.changeYou(clement);
clement.changeMe(alice);
alice.changeComplete(bruno);
bruno.name = "grincheux";
System.out.println(alice + "\t" + bruno + "\t" + clement);
```

Q3. Observez les méthodes `getMe()` et `getLikeMe()`. À quoi servent-elles ? Qu'est-ce qui les différencient ? Qu'affiche le programme suivant ?

```
System.out.println(alice + "\t" + bruno + "\t" + clement);
alice = bruno.getMe();
clement = bruno.getLikeMe();
bruno.name = "grincheux";
System.out.println(alice + "\t" + bruno + "\t" + clement);
```

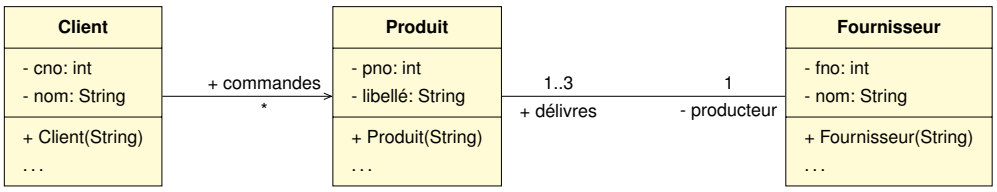
Q4. Observez le reste du code. Quelles sont les méthodes mal écrites ? Qu'affiche le programme suivant ?

```
int otherAge = 404; String otherName = "grincheux";
System.out.println(alice + "\t" + bruno + "\t" + clement);
alice.changeMyAge(otherAge);
alice.changeAName(otherName);
bruno.changeYourAge(otherAge);
alice.changeAnotherName(otherName);
System.out.println(alice + "\t" + bruno + "\t" + clement + "\t" + otherAge+otherName);
otherName = bruno.changeYourName(otherName);
System.out.println(alice + "\t" + bruno + "\t" + clement + "\t" + otherAge+otherName);
```

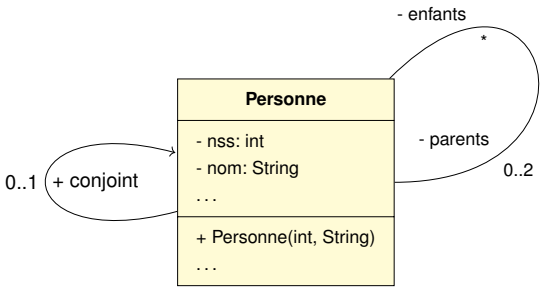
Exercice 2 : Interprétation d'associations UML

À l'image des *Modèles Conceptuels de Données (MCD)* que vous avez vu, les diagrammes UML doivent être interprétés pour connaître les caractéristiques des objets à implémenter.

Q1. Interprétez le diagramme suivant en diagramme d'objets :



Q2. Interprétez le diagramme suivant en diagramme d'objets :



Exercice 3 : Jeu de carte

On dispose d'un type objet `Card` qui représente une carte à jouer. Les couleurs et les rangs sont codés par des chiffres de la manière suivante :

0 : ♣ Club	0 : 7	4 : Jack
1 : ♢ Diamond	1 : 8	5 : Queen
2 : ♡ Heart	2 : 9	6 : King
3 : ♠ Spade	3 : 10	7 : Ace

La représentation UML correspondante est :

Les spécifications de la classe `Card` sont les suivantes :

Class `Card`

	`Card(int c, int rg)` Crée une carte d'une couleur et d'un rang donnés.
`int`	`getColor()` Retourne la couleur de la carte.
`int`	`getRank()` Retourne le rang de la carte.
`int`	`compareRank(Card c)` Retourne : 0 si les deux cartes sont de même rang ; <0 si le rang est inférieur à celui de la carte passée en paramètre ; >0 si le rang est supérieur à celui de la carte passée en paramètre.
`int`	`compareColor(Card c)` Retourne : 0 si les deux cartes sont de même couleur ; <0 si la couleur est inférieure à celle de la carte passée en paramètre ; >0 si la couleur est supérieure à celle de la carte passée en paramètre.
`boolean`	`isBefore(Card c)` Compare le rang (puis la couleur en cas d'égalité), retourne `true` si la carte courante précède la carte passée en paramètre ou `false` sinon.
`boolean`	`equals(Card c)` Retourne `true` si les deux cartes sont identiques, `false` sinon.
`String`	`toString()` Retourne la carte sous la forme d'une chaîne : "`Club 7`" ou "7♣".

Q1. Listez les attributs de la classe `Card`. Décrivez son constructeur et détaillez la méthode `equals`.

Q2. Ajoutez les méthodes `compareRank`, `compareColor`, et `isBefore`.

Q3. Écrivez une classe `UseCard` contenant une méthode `main` qui crée et affiche les 32 cartes d'un jeu complet.

Q4. Modifiez cette classe pour générer deux cartes aléatoirement et les afficher dans l'ordre croissant (grâce à la méthode `isBefore`).

Q5. Modifiez encore cette classe pour qu'elle génère un nombre nb de cartes aléatoires, où nb correspond à la première chaîne du tableau d'arguments `args` de la méthode `main`. Le programme doit afficher les cartes une par une, puis affiche la carte la plus forte.