HE2B ESI

Analyse 1

ANA2

Geneviève Cuvelier (CUV)
Christine Leignel (CLG)
Thibaut Nicodème (TNI)
Pantélis Matsos (PMA)

v1.2

2020-2021

Où en sommes-nous ?

1. Qu’est-ce que l’analyse?
2. Diagramme d’activités
3. Les classes et objets
4. Les associations 1-1 et 1-N
5. Les associations N-N
6. Les compositions et énumérations
7. Les classes associations
8. L’héritage
9. Les interfaces

Classes « types de données »

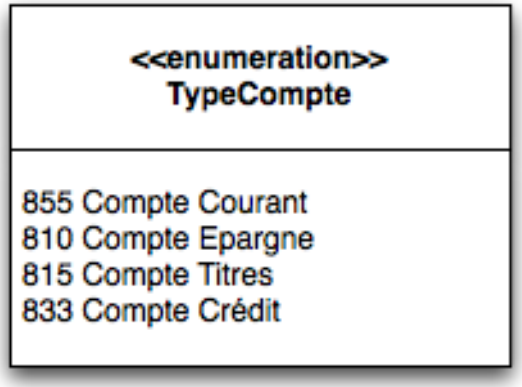
UML propose les types « standard » de données pour les attributs : entiers, booléens, date, ...

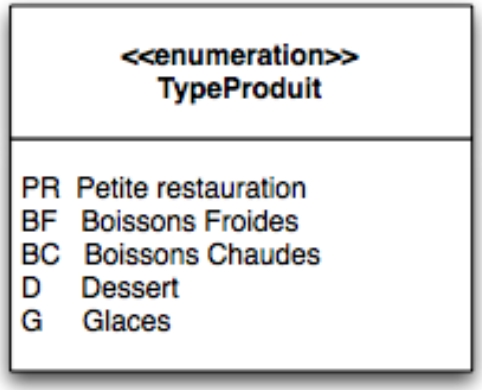
- On peut créer des données de type particulier qui seront reprises dans des classes.
- Notation: au-dessus du nom de la classe *particulière*, on place le type de données entre << >>

Classes « types de données »

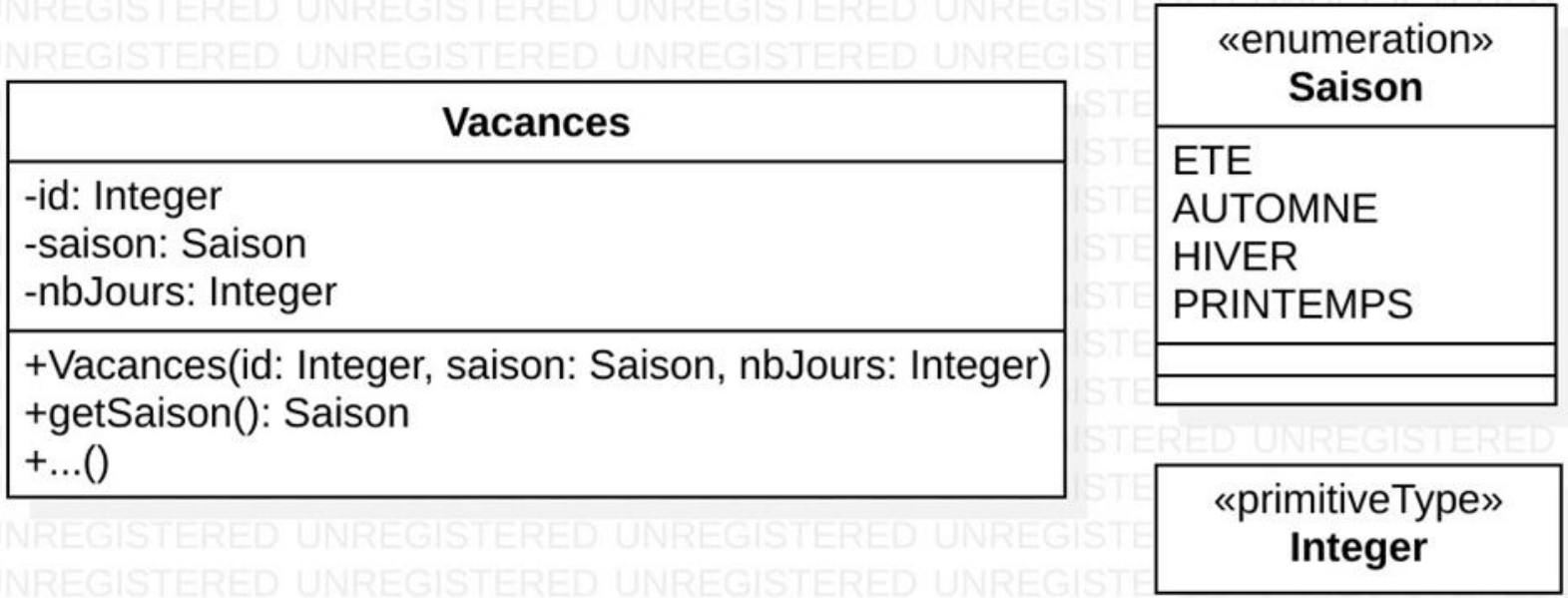
- Classe <<énumération>>
  - deux attributs: un code et un libellé (nom).
  - listes connues des utilisateurs.
  - une instance correspond à une valeur de la liste.
- Classe <<structure>>
  - modélisation d’un attribut décomposable (ou composé)
- Ces classes ne peuvent pas avoir d’associations

Classes «enumeration»: exemples





Classes «enumeration»: exemples

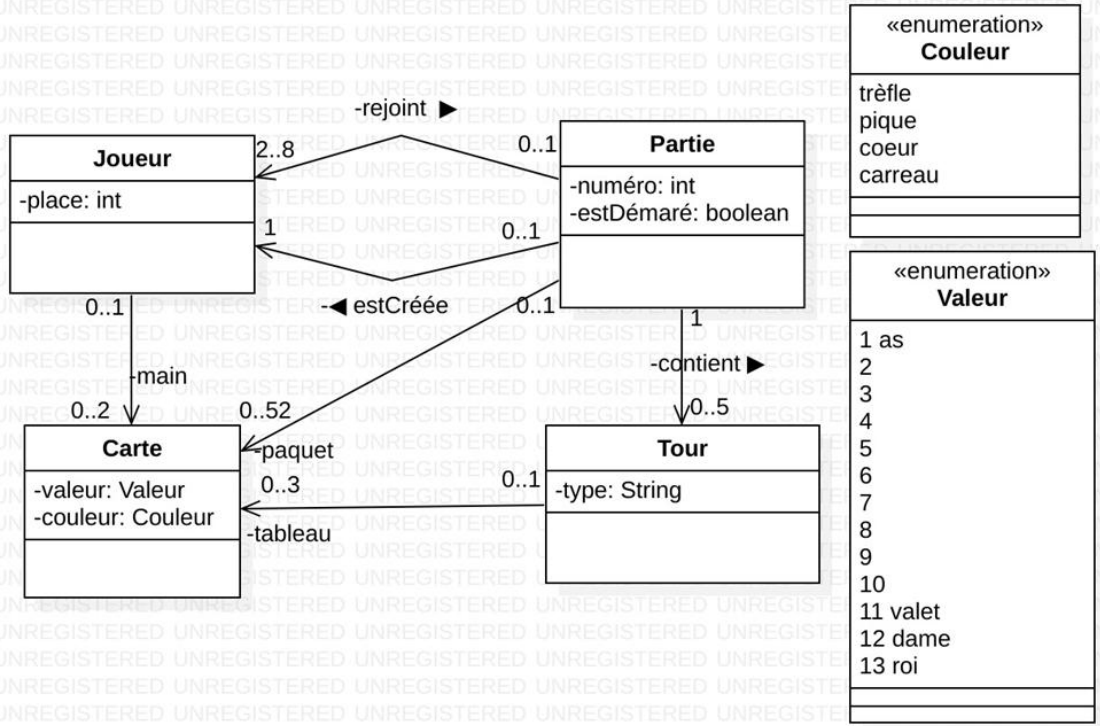


Énumération - Exercice 1

Dans le diagramme du Poker (multiplicité, exercice 4), remplacez les types *standard* par des énumérations lorsque cela fait sens ?

Écrivez leurs code Java.

Énumération - Solution 1



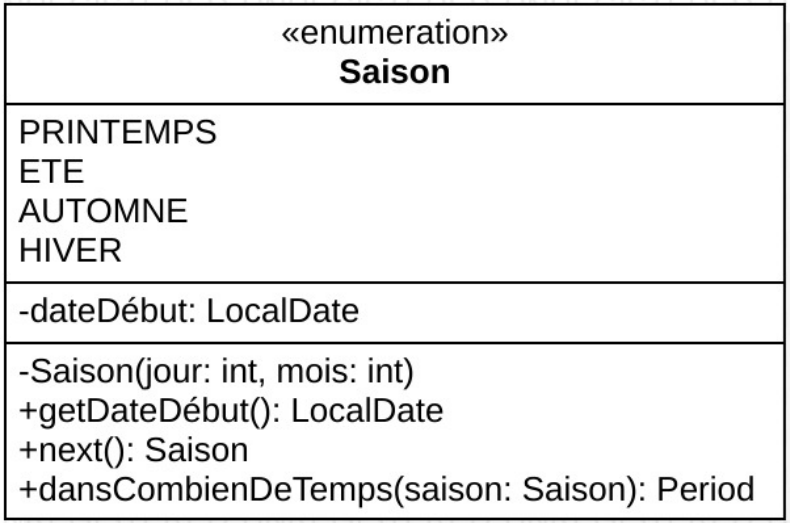
Énumération – Exercice 2

Traduisez le code de l'enum *Saison*

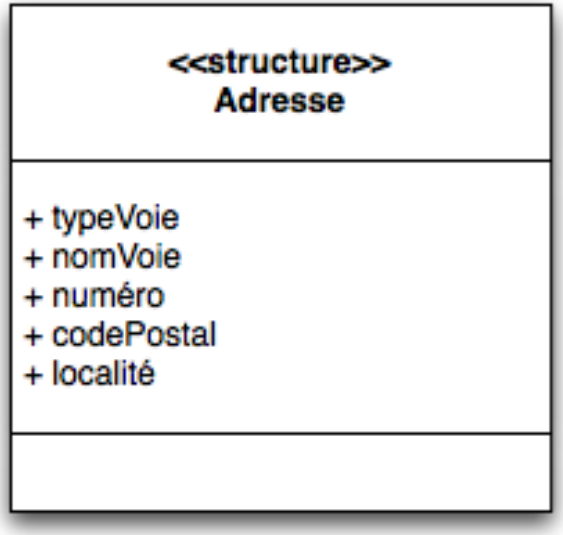
en diagramme de classe.

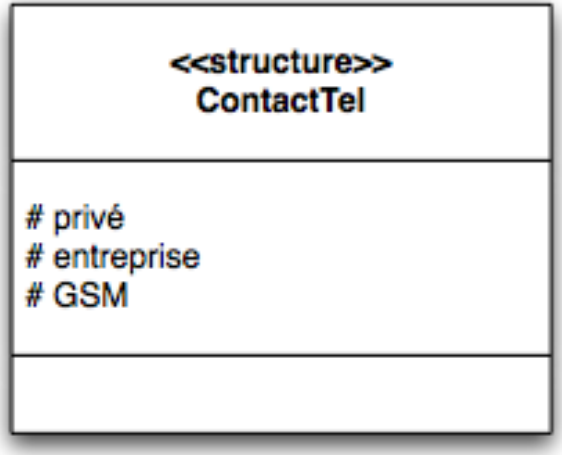
```
public enum Saison {
    PRINTEMPS(21, 3),
    ÉTÉ(22, 6),
    AUTOMNE(23, 9),
    HIVER(21, 12);

    private final LocalDate dateDébut;

    private Saison(int jour, int mois){
        LocalDate now = LocalDate.now();
        this.dateDébut = LocalDate.of(now.getYear(), mois, jour);
    }

    public LocalDate getDateDébut(){
        return this.dateDébut;
    }

    public Saison next(){
        Saison[] saisons = this.values();
        return saisons[(this.ordinal() + 1) % saisons.length];
    }
}
```

Énumération – Solution 2



Classes «structure»: exemples





Structure – Exercice 1

Écrivez le code Java de la structure Adresse du *slide* précédent.

Utiliser des classes « structure » en Java vous semble-t-il une bonne idée ?

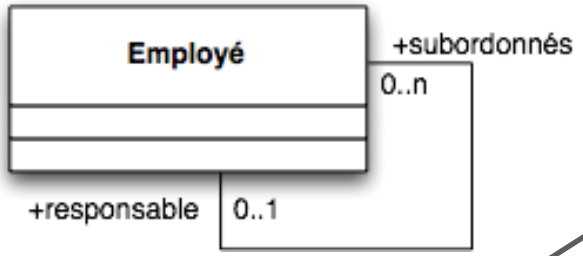
Structure - Solution 1

```
public class Adresse {
   public TypeVoie typeVoie;
   public String nomVoie;
   public int numero;
   public int codePostal;
   public String localite;
}

public enum TypeVoie {
   Avenue, Rue, Chaussée, Boulevard
}
```

Association réflexive

- Association qui part et arrive sur une même classe.
- Elle signifie donc qu’il existe des liens entre objets d’une classe.

Association réflexive – Exemple



Association réflexive – Exercice 1

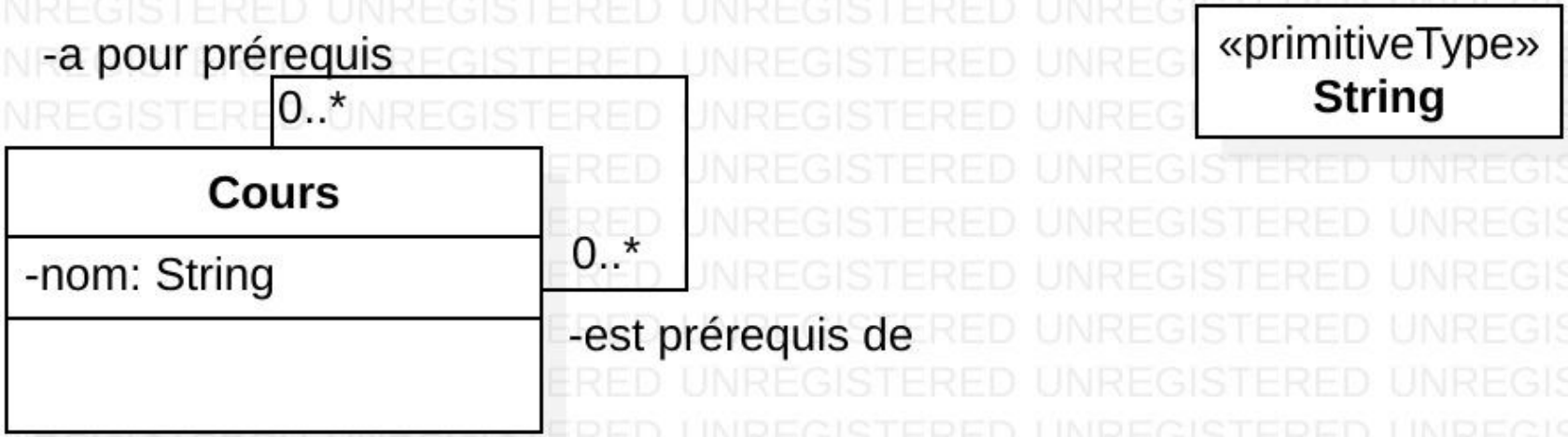
Dessinez le diagramme de classe représentant l'association entre des cours et leurs prérequis.

Ensuite écrivez le code Java et créez des objets *cours* tels que :

- DEV2 dépend de DEV1,
- MICL dépend de DEV1 et INO,
- le stage dépend de tous les cours.

Finalement dessinez le diagramme d'objets représentant les différents cours créés.

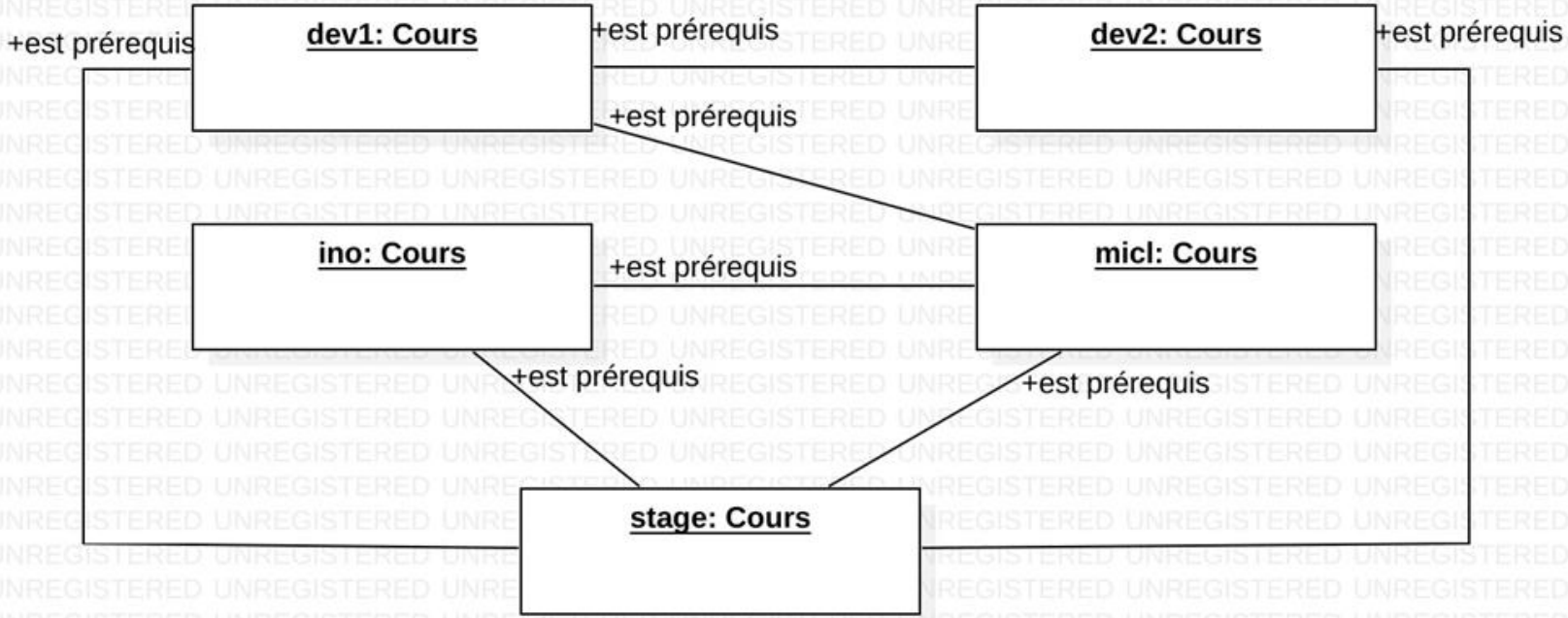
Association réflexive - Solution 1



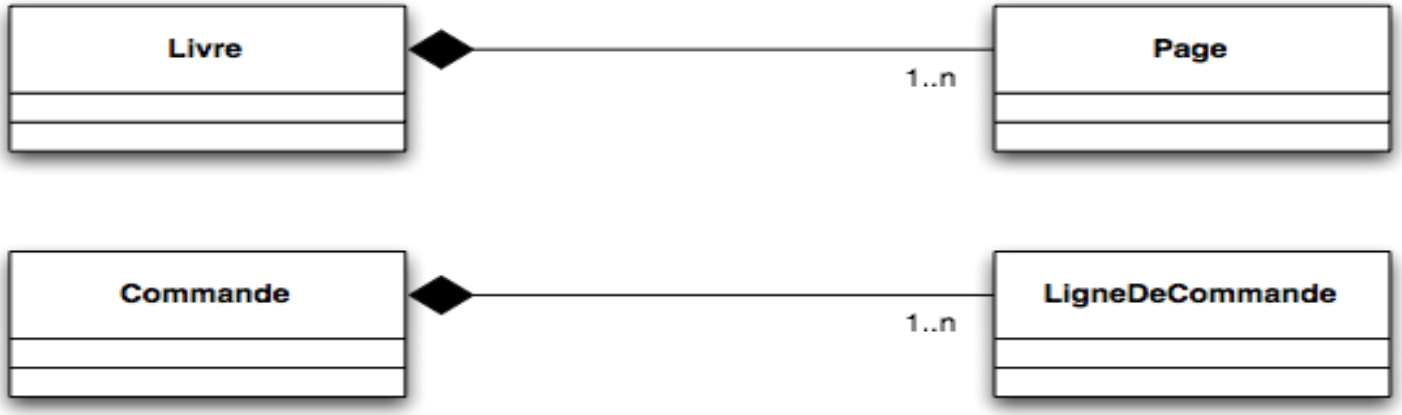
Association réflexive - Solution 1

```java
import java.util.List;
import java.util.Arrays;

public class Cours {
   private final String nom;
   private final List<Cours> prerequis;

   public Cours(String nom, Cours ...prerequis) {
       this.nom = nom;
       this.prerequis = Arrays.asList(prerequis);
   }

   public static void main(String[] args) {
       Cours dev1 = new Cours("dev1");
       Cours dev2 = new Cours("dev2", dev1);
       Cours ino = new Cours("ino");
       Cours micl = new Cours("micl", dev1, ino);
       Cours stage = new Cours("stage", dev1, dev2, ino, micl);
   }
}
```

Association réflexive - Solution 1



Composition

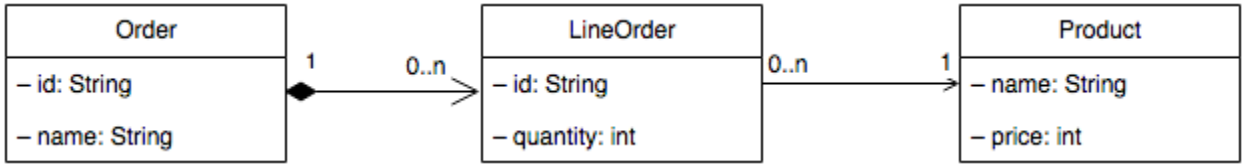


- Ajoute une sémantique : “est composé de” / “fait partie de”
- Contraintes liées à la composition :
  - Un objet *composant* ne peut être que dans 1 seul objet *composite*
  - Un objet *composant* n’existe pas sans son objet *composite*
  - Si un objet composite est détruit, ses composants aussi

Composition – Exercice 1

Traduisez le diagramme UML suivant en code Java.

Écrivez également un petit *main* pour illustrer l'utilisation de vos classes.



Composition - Solution 1

```
public class Product {
    private final String name;
    private final int price;

    public Product(String name, int price) {
        this.name = name;
        this.price = price;
    }

    public int getPrice() {
        return price;
    }

    public String getName() {
        return name;
    }
}
```

```
public class LineOrder {
    private final String id;
    private final int quantity;
    private final Product product;

    public LineOrder(String id, int quantity, Product p) {
        this.id = id;
        this.quantity = quantity;
        this.product = p;
    }

    public String getId() {
        return id;
    }

    public int getQuantity() {
        return quantity;
    }

    public Product getProduct() {
        return product;
    }
}
```

Composition - Solution 1, suite

```
public class Order {
   private String id;
   private String name;
   private List<LineOrder> lineOrders;

   public Order(String id, String name) {
       this.id = id;
       this.name = name;
       this.lineOrders = new ArrayList<LineOrder>();
   }

   public String getId() {
       return id;
   }

   public void addItem(String id, int quantity, Product p) {
       this.lineOrders.add(new LineOrder(id, quantity, p));
   }
}
```

Composition - Solution 2, suite

```
public class Test {
   public static void main(String[] args) {
       Product p1 = new Product("sauce tomate", 3);
       Product p2 = new Product("papier toilette", 5);

       Order o = new Order("12-ABC", "commande spéciale");
       o.addItem("XYZ-123", 15, p1);
       o.addItem("DEF-456", 10, p2);

       /* Il est possible de créer un `LineOrder` correspondant à aucun `Order`.
          Ceci est contraire aux multiplicités du diagramme.
          Un moyen de résoudre ceci est d'utiliser une `inner class`.
          Mais ceci ne fait pas partie de la matière de DEV2. */
       LineOrder lineOrderSansOrder = new LineOrder("KLM-789", 100, p2);
   }
}
```