Interfaces, Classes Abstraites - Génériques

1 Les gaulois

Nous restons dans le même thème que pour le TP précédent : sur le marché d'un village *Gaulois*. Dans ce TP nous allons commencer par cerner les limites de la généricité sur l'utilisation d'objets placés dans des tableaux par rapport à nos connaissances en ILU2, puis nous ferons une nouvelle implémentation du village afin de prendre en compte toutes les abstractions étudiées cette année : interface avec méthode par défaut, classe et méthode abstraite, classe et méthode générique et classe anonyme.

2 Mise en place de l'environnement

Créer un nouveau projet, comportant 3 paquetages :

- personnages, dans lequel vous copierez les classes Personnage et Gaulois
- villageGaulois, dans lequel vous copierez la classe Etal,
- scenario, dans lequel vous créez la classe ScenarioTest, comportant un main.

Créer un dernier paquetage *produit* contenant la classe *Produit*. Un produit a un nom, et une unité. Les unités sont l'une des quantités suivantes : (gramme, kilogramme, litre, centilitre, millilitre ou par pièce). Elles devront s'afficher selon le système international (g, kg, l, cl, ml, piece).
La classe Produit possède un getteur sur le nom. Elle possède aussi une méthode qui décrit le produit *decrireProduit*. Les descriptions des produits sont très différentes selon les produits concernés (nous n'aurons pas du tout la même description entre du poisson et une lyre).

Pour le TP nous allons spécifier deux produits : du poisson et du sanglier.
Le poisson est un produit qui possède :

- une date de pêche (une chaîne de caractères),
- un nom, en tant que produit, tous les poissons ont le même nom : "poisson".

Il est vendu à la pièce.

Exemple de description : "poisson pêchés mardi."

Le sanglier est un produit qui possède :

- un poids,
- un chasseur (le gaulois qui a tué le sanglier),
- un nom, en tant que produit, tous les sangliers ont le même nom : "sanglier".

Il est vendu au kilo.

Exemple de description : "sanglier de 15 kg chassé par Obélix."

Réfléchir SUR PAPIER à une modélisation des produits en utilisant un diagramme de classe en utilisant énumération et abstractions (classe / méthode abstraite). Faire valider par votre encadrant de TP.

Tableau et généricité - les limites

1. Les étals - essai 1

Les étals vont pouvoir être personnalisés à leur création pour un type de produit spécifique ou non. Par exemple un étal peut être construit spécifiquement :

- pour vendre du poisson, il aura un revêtement en céramique afin de pouvoir le nettoyer facilement,
- pour la vente de sanglier, il aura un billot intégré afin de pouvoir effectuer la découpe de la viande,
- ...

D'autres étals sont construits sans spécificités pour accueillir n'importe quel vendeur.

Travail à effectuer : modifier dans la classe *Etal* le type de l'attribut *produit* et celui de son getteur afin d'utiliser un type générique P sous-type de Produit.

Observation : dans la classe ScenarioTest écrire l'instruction

```
Etal[] marche = new Etal[3];
Etal<Sanglier> etalSanglier = new Etal<>();
Etal<Poisson> etalPoisson = new Etal<>();
marche[0] = etalSanglier;
marche[1] = etalPoisson;
```

Cette instruction compile mais un warning apparaît : "*Etal is a raw type. References to generic type Etal<P> should be parameterized*".
La classe *Etal* est générique mais un tableau ne peut pas être créé avec un type générique ~~Etal<Sanglier> marche = new Etal<>[3];~~.

2. Interface IEtals générique - essai 2

Nous allons tenter d'utiliser une interface afin de créer un tableau d'étals.

Travail à effectuer : Extraire l'interface *IEtals* de la classe *Etal* :

- placer vous dans l'explorateur de paquetage (Package Explorer)
- cliquer droit sur la classe *Etal* puis
  Refactor > Extract Interface

- Nommer l'interface *IEtal*,
- Sélectionner l'ensemble des méthodes
- générer l'interface

Observations : *IEtal* est extrait comme étant un type générique :

public interface IEtal<P extends Produit>

Travail à effectuer : Dans la classe ScenarioTest typer le tableau d'étal avec l'interface :

IEtal<Produit>[] marche = new IEtal[3];

Observation : l'instruction compile, mais un warning apparaît.
Type safety: The expression of type IEtal[] needs unchecked conversion to conform to IEtal<Sanglier>[]

Travail à effectuer :

- Créer un étal de sanglier.
- Modifier la déclaration de la variable "*marché*" pour préciser que ses étals acceptent uniquement des sangliers.

Observations :

Peut-on placer l'étal dans le marché ?

Conclusion : par rapport au connaissance de l'ILU2 on ne peut pas préciser que le marché peut prendre un étal de *Produit* ou de ses classes filles *Sanglier* ou *Poisson*. Pour l'instant le marché ne peut être composé que d'étals de sangliers ou d'étals de poissons mais pas des deux. Nous pourrons débloquer cette utilisation en ILU3 avec le typage des génériques.

3. Interface IEtals non générique - essai 3

Nous allons enlever la généricité au niveau de l'interface afin de pouvoir créer un marché avec des étals de sangliers et des étals de poissons

Travail à effectuer :

Dans l'interface *IEtal* :

- enlever la généricité au niveau de l'interface,
- remplacer le type générique *P* par le type *Produit*.

Dans la classe *Etal* :

- modifier les types *P*, dans les méthodes implémentant celle de l'interface, par *Produit*,
- dans la méthode *occuperEtal* il faut caster le *produit* de type *Produit* en type *P* pour affecter le paramètre d'entrée à l'attribut *produit* (ce qui n'est pas très beau).

Dans la classe *ScenarioTest* : Enlever la généricité sur l'interface.

```
IEtal[] etals = new IEtal[3];
Etal<Sanglier> etalSanglier = new Etal<>();
etals[0] = etalSanglier;
```

Observation : L'étal placé dans la case 0 est du type *Sanglier*. Or vous pouvez occuper cet étal par un vendeur de poisson :

```
etals[0].occuperEtal(new Gaulois("Ordralfabétix", 12), new Poisson("lundi"), 10);
```

Pourquoi ?

Dans la classe *Etal*, la méthode *occuperEtal* :

```
public void occuperEtal(Gaulois vendeur, Produit produit, int quantite) {
    this.produit = (P) produit;
    ...
}
```

a. Le paramètre d'entrée est un *Poisson* sous-type de *Produit* donc l'appel à *occuperEtal* est valide.
b. le cast *P* est traduit par le compilateur en cast (*Produit*) qui est inutile car *this.produit* et *produit* sont typés *Produit* par le compilateur.
c. Au final il n'y a donc aucun contrôle de type et un poisson sera stocké là où un sanglier était attendu. Ainsi on peut vendre du poisson sur un étal de sanglier.

Conclusion :
on ne peut pas imposer à un tableau de contenir un type paramètre de généricité.

Nouveau sujet

Aller sur Github et récupérer le projet ILU2-POO-TP4. Récupérer de la partie précédente le paquetage produits avec les classes Produit, Poisson, Sanglier et l'énumération.

Nouvelle classe Etal

Travail à effectuer :

1. Extraire de la classe *Produit* l'interface *IProduit*.
2. La classe *Produit* doit implémenter *IProduit*. SonarLint vous demande d'effectuer une modification. La comprendre et procéder à la modification.

3. Dans le paquetage *villagegaulois* créer l'interface *IEtal* :

```
public interface IEtal {
    Gaulois getVendeur();
    int contientProduit(String produit, int quantiteSouhaitee);
    int acheterProduit(int quantiteSouhaitee);
    String etatEtal();
}
```

4. Dans le paquetage *villagegaulois prendre* connaissance de la classe *Etal*.
   a. cette classe possède 4 attributs :
      - un vendeur *vendeur* : un gaulois,
      - un produit de type String,
      - deux entiers indiquant le nombre de produit en début et en fin de marché,
      - un boolean indiquant si l'étal est occupé.

      Ajouter le prix du produit *prix* : un entier les "sous" n'avait pas de centimes.

      On souhaite maintenant ne pas vendre un produit de type String, mais plusieurs produits d'un même type issu de IProduit. On ne pourra vendre que des sangliers, que des poissons ou que des fleurs... sur un même étal. Supprimer l'attribut *produit* pour le remplacer par :
      - un tableau de produit *produits* : du type générique de l'étal, pour restreindre le générique utiliser l'interface *IProduit*,
      - un entier *nbProduit* : la quantité de produits sur l'étal (initialisé à 0).

   b. Ajouter la méthode *installerVendeur(Gaulois vendeur, P[] produit, int prix)* qui initialise les attributs avec les paramètres d'entrée de la méthode, le nombre de produits dans le tableau est initialisé à la taille du tableau)

      Dans la classe *ScenarioTest* :
      - Créer des gaulois :

```
Gaulois ordralfabetix = new Gaulois("Ordralfabétix",9);
        Gaulois obelix = new Gaulois("Obélix",20);
        Gaulois asterix = new Gaulois("Asterix", 6);
```

      - Créer les tableaux de produits :

```
Sanglier sanglier1 = new Sanglier(2000, obelix);
Sanglier sanglier2 = new Sanglier(1500, obelix);
Sanglier sanglier3 = new Sanglier(1000, asterix);
Sanglier sanglier4 = new Sanglier(500, asterix);

Sanglier[] sangliersObelix = {sanglier1, sanglier2};
```

```
Sanglier[] sangliersAsterix = {sanglier3, sanglier4};

Poisson poisson1 = new Poisson("lundi");
Poisson[] poissons = {poisson1};
```

- Créer le marché :
  Créer un marché (variable marche) comme un tableau de l'interface IEtal pouvant contenir 3 étals.
  Créer 2 étals de sanglier et 1 étal de poisson.
  Placer ces étals dans le marché.

- Installer les vendeurs sur le marché :
  Le vendeur Obélix vend les sangliers qu'il a chassé au prix de 8 sous.
  Le vendeur Astérix vend les sangliers qu'il a chassé au prix de 10 sous.
  Le vendeur Ordralfabétix vend le poisson qu'il a péché au prix de 7 sous.

c. Modifier la signature de la classe *Etal* pour qu'elle implémente l'interface *IEtal*. Compléter la classe *Etal* en conséquence.

Aide :

```
@Override
public int contientProduit(String produit, int quantiteSouhaitee) {
    int quantiteAVendre = 0;
    if (nbProduit != 0 && this.produits[0].getNom().equals(produit)) {
      if (nbProduit >= quantiteSouhaitee) {
        quantiteAVendre = quantiteSouhaitee;
      } else {
        quantiteAVendre = nbProduit;
      }
    }
    return quantiteAVendre;
}

@Override
public int acheterProduit(int quantiteSouhaite) {
    int prixPaye = 0;
    for (int i = nbProduit - 1; i > nbProduit - quantiteSouhaite - 1 || i > 1; i--) {
      prixPaye += produits[i].calculerPrix(prix);      //question 3.d
    }
    if (nbProduit >= quantiteSouhaite) {
      nbProduit -= quantiteSouhaite;
    } else {
      nbProduit = 0;
    }
    return prixPaye;
```

```
}

@Override
public String etatEtal() {
    StringBuilder chaine = new StringBuilder(vendeur.getNom());
    if (nbProduit > 0) {
      chaine.append(" vend ");
      chaine.append(nbProduit + " produits :");
      for (int i = 0; i < nbProduit; i++) {
          chaine.append("\n- " + produits[i].decrireProduit());
      }
    } else {
      chaine.append(" n'a plus rien à vendre.");
    }
    chaine.append("\n");
    return chaine.toString();
}
```

d. La méthode *acheterProduit* appelle, sur un objet de type *IProduit*, la méthode calculerPrix(prix). Cette méthode permet de récupérer le prix du produit acheté en donnant en paramètre d'entrée le prix fixé par le vendeur au niveau de son étal.
La plupart du temps ce prix ne change pas : un marchand qui vend des salades peut fixer le prix à 5 sous la salade. Si on applique la méthode à la salade alors *salade.calculerPrix(5)* retournera 5. L'acheteur devra dépenser 5 sous.
Par contre certains prix peuvent changer selon le produit acheté : un marchand qui vend des fraises peut fixer un prix à 9 sous le kg. Si on applique la méthode à un sachet de 500g *sachetFraise.calculerPrix(9)* retournera 4. L'acheteur devra dépenser 4 sous.

Trouver la meilleure façon d'implémenter cette méthode.

e. Dans la classe *ScenarioTest* vérifier que toutes les méthodes sont correctes :
1. afficher les étals du marché (utilisation de la méthode etatEtal de la classe Etal),
2. acheter 3 sangliers (utilisation de la méthode static *acheterProduit*)
3. afficher de nouveau les étals du marché

Sortie Attendue :

```
Asterix vend 2 produits :
- sanglier de 1000 kg chassé par Asterix.
- sanglier de 500 kg chassé par Asterix.
```

Obélix vend 2 produits :
- sanglier de 2000 kg chassé par Obélix.
- sanglier de 1500 kg chassé par Obélix.

Ordralfabétix vend 1 produits :
- poisson pêché lundi.

A l'étal n° 1, j'achète 2 sangliers et je paye 12 sous.
A l'étal n° 2, j'achète 1 sanglier et je paye 15 sous.
Je voulais 3 sangliers, j'en ai acheté 3.

Asterix n'a plus rien à vendre.

Obélix vend 1 produits :
- sanglier de 2000 kg chassé par Obélix.

Ordralfabétix vend 1 produits :
- poisson pêché lundi.

Un village - Classe Anonyme

Pour cette dernière partie nous allons créer l'objet *village* à partir d'une classe anonyme afin de créer un village particulier. Dans la classe *Scenario* créer au niveau du tag TODO la classe interne *Village* implémentée à partir de l'interface *IVillage* dans le paquetage *village* :

```
public interface IVillage {
        <P extends Produit> boolean installerVendeur(Etal<P> etal,
                Gaulois vendeur, P[] produit, int prix);

        void acheterProduit(String produit, int quantiteSouhaitee);
}
```

Redéfinir la méthode toString() dans la classe anonyme afin qu'elle affiche les étals du marché comme dans l'exemple ci-dessous.

Sortie attendue :

Astérix vend 2 produits :
- sanglier de 1000 kg chassé par Astérix.
- sanglier de 500 kg chassé par Astérix.

Obélix vend 2 produits :
- sanglier de 2000 kg chassé par Obélix.
- sanglier de 1500 kg chassé par Obélix.

Ordralfabétix vend 1 produits :
- poisson pêché lundi.

A l'étal n° 1, j'achète 2 sangliers et je paye 15 sous.
A l'étal n° 2, j'achète 1 sanglier et je paye 12 sous.
Je voulais 3 sangliers, j'en ai acheté 3.

Astérix n'a plus rien à vendre.

Obélix vend 1 produits :
- sanglier de 2000 kg chassé par Obélix.

Ordralfabétix vend 1 produits :
- poisson pêché lundi.