Conception orientée objet

Application du cours 2 : Musée

Le sujet permet de nous entrainer à utiliser :
- des interfaces notamment en utilisant le design pattern « bridge »,
- la création, la levée et le traitement d'exception.

I. Environnement du problème

Comme pour la première application de cours nous travaillons sur la vie des habitants de l'Armoricae d'après l'œuvre de René Goscinny, Albert Uderzo et Jean-Yves Ferri : « Astérix ». Nous étudierons dans cette application le stockage des trophées des Gaulois dans leur musée « Museum ».

II. Les classes vues dans le cours

1. Package musee

Interface GestionTrophee

```
public interface GestionTrophee {
  void ajouterTrophee(Gaulois proprietaire, Equipement trophee);
  String tousLesTrophees();
  String lesTrophees(Gaulois proprietaire);
}
```

Classe Musee

```
public class Musee {
  private String nom;
  private int tarif;
  private GestionTrophee gestionnaireTrophee;

  public Musee(String nom, GestionTrophee gestionnaireTrophee) {
    this.nom = nom;
    this.gestionnaireTrophee = gestionnaireTrophee;
  }
  public String getNom() {
    return nom;
  }
  public void setTarif(int tarif) {
    this.tarif = tarif;
  }
  public int getTarif() {
    return tarif;
  }
  public void ajouterTrophee(Gaulois proprietaire,
                             Equipement trophee) {
    gestionnaireTrophee.ajouterTrophee(proprietaire, trophee);
  }
  public String tousLesTrophees() {
    return gestionnaireTrophee.tousLesTrophees();
  }
```

```
  public String lesTrophees(Gaulois proprietaire) {
    return gestionnaireTrophee.lesTrophees(proprietaire);
  }
}
```

Classe GoudurixGestion

```
public class GoudurixGestion implements GestionTrophee {
  private Gaulois[] proprietaires = new Gaulois[30];
  private Equipement[] trophees = new Equipement[30];
  private int nombreDeTrophee = 0;

  public void ajouterTrophee(Gaulois proprietaire, Equipement trophee){
    proprietaires[nombreDeTrophee] = proprietaire;
    trophees[nombreDeTrophee] = trophee;
    nombreDeTrophee++;
  }

  public String lesTrophees(Gaulois proprietaire) {
    String leTrophee = "Le trophée de " + proprietaire.getNom() + " est ";
    int indiceProprietaire = 0;
    for (int i = 0; i < nombreDeTrophee; i++) {
      if (proprietaire.equals(proprietaires[i])) {
        indiceProprietaire = i;
      }
    }
    leTrophee += trophees[indiceProprietaire];
    return leTrophee;
  }

  public String tousLesTrophees() {
    String tousLesTrophees = "Tous les trophées du musée sont :\n";
    for (int i = 0; i < nombreDeTrophee; i++) {
      tousLesTrophees += "- " + trophees[i] + "\n";
    }
    return tousLesTrophees;
  }
}
```

Classe RenseignementTrophee

```
public class RenseignementTrophee {
  private Gaulois proprietaire;
  private Equipement trophee;

  public RenseignementTrophee(Gaulois proprietaire,
                              Equipement trophee) {
    this.proprietaire = proprietaire;
    this.trophee = trophee;
  }

  public Gaulois getProprietaire() {
    return proprietaire;
  }

  public Equipement getTrophee() {
    return trophee;
  }
}
```

Classe KeskonrixGestion

```
public class KeskonrixGestion implements GestionTrophee {
  private RenseignementTrophee[] trophees =
    new RenseignementTrophee[30];
  private int nombreDeTrophee = 0;

  public void ajouterTrophee(Gaulois proprietaire,
        Equipement trophee) {
    trophees[nombreDeTrophee] =
      new RenseignementTrophee(proprietaire, trophee);
    nombreDeTrophee++;
  }

  public String lesTrophees(Gaulois proprietaire) {
    String tousLesTrophees = "Les trophées de " +
      proprietaire.getNom() + " sont :\n";
    for (int i = 0; i < nombreDeTrophee; i++) {
      Gaulois proprietaireTrophee = trophees[i].getProprietaire();
      if (proprietaire.equals(proprietaireTrophee)) {
        Equipement typeEquipement = trophees[i].getTrophee();
        tousLesTrophees += "- " + typeEquipement + "\n";
      }
    }
    return tousLesTrophees ;
  }

  public String tousLesTrophees() {
    String tousLesTrophees = "Tous les trophées du musée sont :\n";
    for (int i = 0; i < nombreDeTrophee; i++) {
      Equipement typeEquipement = trophees[i].getTypeTrophee();
      tousLesTrophees += "- " + typeEquipement + "\n";
    }
    return tousLesTrophees;
  }
}
```

III. Enoncé du TD

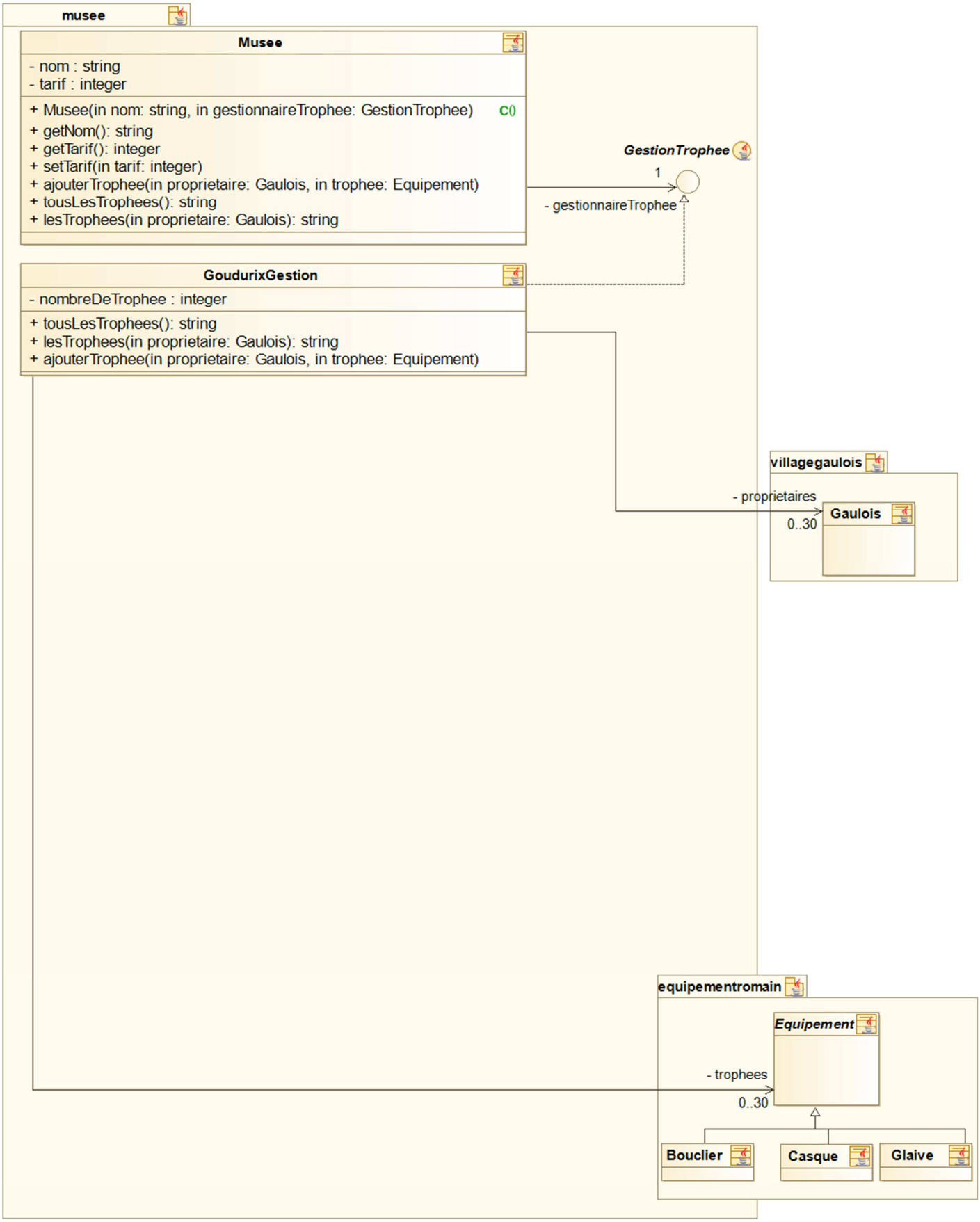
1. Diagramme de classes de la solution de Keskonrix

Les gaulois souhaitent pouvoir consulter leurs trophées (équipement d'un soldat romain). Il faut donc qu'une entité rende les services suivants :

- ajouter un trophée,
- récupérer l'ensemble des trophées du village,
- récupérer l'ensemble des trophées apportés par un gaulois en particulier.

Ces services sont repris dans l'interface « *GestionTrophee* ».
Keskonrix, un aspirant à devenir informaticien (non confirmé du tout), a apporté l'implémentation «KeskonrixGestion» à l'interface. Il a créé la classe « RenseignementTrophee » lui permettant d'associer un propriétaire à un trophée.

Modifier le diagramme de classe ci-dessous pour prendre en compte la solution de Keskonrix.



Modifier si besoin la classe « Test » ci-dessous.

```
public class Test {
  public static void main(String[] args) {
    Gaulois asterix = new Gaulois("Astérix");
    Gaulois obelix = new Gaulois("Obélix");
    Gaulois abraracourcix = new Gaulois("Abraracourcix");

    Bouclier bouclierMordicus = new Bouclier("bon état");
    Casque casqueAerobus = new Casque("cabossé", "fer");
    Glaive glaiveCornedurus = new Glaive("cassé");
    Glaive glaiveAerobus = new Glaive("tordu");
    Casque casqueHumerus = new Casque("bon état", "fer");

    GestionTrophee gestionnaireTrophees = new GoudurixGestion();
    Musee musee = new Musee("Museum", gestionnaireTrophees);
    musee.ajouterTrophee(asterix, bouclierMordicus);
    musee.ajouterTrophee(asterix, casqueAerobus);
    musee.ajouterTrophee(asterix, glaiveCornedurus);
    musee.ajouterTrophee(obelix, glaiveAerobus);
    musee.ajouterTrophee(abraracourcix, casqueHumerus);
    System.out.println("Le musé " + musee.getNom()
      + " est ouvert !\n");
    System.out.println(musee.tousLesTrophees());
    System.out.println(musee.lesTrophees(asterix));
  }
}
```

Modifier si besoin l'extrait de la classe « Musee » ci-dessous.

```
public class Musee {
  private String nom;
  private int tarif;
  private GestionTrophee gestionnaireTrophee;

  public Musee(String nom, GestionTrophee gestionnaireTrophee) {
    this.nom = nom;
    this.gestionnaireTrophee = gestionnaireTrophee;
  }

  …
}
```

2. Les exceptions

Dans le village gaulois, on organise régulièrement des festins. Pour les financer, on propose des activités payantes.
Créer l'interface « ActivitePayante » ayant deux méthodes : *getTarif* et *setTarif*.

Interface ActivitePayante

Reprendre la classe « Musee » afin qu'elle implémente cette interface.

D'autres activités sont proposées : lancer de menhir, bataille de poissons (pas frais), chasse aux sangliers …
Chacune de ses activités est représentée par une classe qui implémente l'interface « ActivitePayante ».

Nous proposons la classe « Billetterie » qui permet d'acheter des places à une activité donnée.

```
public class Billeterie {
  private int buttin = 0;
  private ActivitePayante[] activites = new ActivitePayante[4];
  private int nbActivite = 0;

  public void ajouterActivite(ActivitePayante activitePayante,
      int prixBillet) {
    activites[nbActivite] = activitePayante;
    activitePayante.setTarif(prixBillet);
  }

  public void acheterBillet(int numeroActivite){
    ActivitePayante activite = activites[numeroActivite];
    buttin += activite.getTarif();
  }

  public int getButtin() {
    return buttin;
  }
}
```

Dans la classe « Billetterie », lorsque l'on achète une place, l'utilisateur fournit le gaulois et le numéro de l'activité, celui-ci ne peut malheureusement correspondre à aucune activité.
Admettons maintenant que l'utilisateur ne puisse acheter des billets que pour activité existante.

Il clique sur un personnage puis sur une activité (l'affichage étant réalisé qu'à partir des activités existantes dans le tableau activites de la classe « Billeterie »). **Il est donc impossible à l'utilisateur de se tromper.**

Donc lorsqu'on appelle la méthode *acheterBillet* le numéro de l'activité est forcément valide par rapport au fonctionnement normal de l'application. Le contraire serait donc une exception.

Laissons apparaitre l'exception.

```
public static void main(String[] args) {
   Billeterie billeterie = new Billeterie();
   billeterie.acheterBillet(1);
}
```

L'exception levée est : java.lang.NullPointerException

Traiter l'exception :

```
public void acheterBillet(int numeroActivite) {
```

```
}
```

Normalement, dans la classe « Billetterie », le tarif de l'activité est affecté lors de son ajout dans le tableau activites (méthode *ajouterActivite*).
Donc lorsqu'on appelle la méthode *acheterBillet* le tarif de l'activité a forcément été initialisé par rapport au fonctionnement normal de l'application. Le contraire serait donc une exception.

Ecrire l'exception personnalisée « TarifNonInitialiserException ».

Lever l'exception personnalisée dans la méthode *getTarif* de la classe « Musee ». Rappel : un entier primitif est initialisé à 0 par défaut.

```
public int getTarif() {
}
```

Modifier la méthode *acheterBillet* de la classe « Billetterie » en conséquence.