Conception orientée objet

Application du cours 4 : Musée et potion magique

Cet énoncé vient en appui des diapositives du Cours.

I. Le parcours des collections

Supprimer le deuxième et troisième élément de la collection suivante :

```
List<Character> caracteres =  new ArrayList<>();
Collections.addAll(caracteres, 'r', 'a', 'u', 't');
Iterator<Character> iter = caracteres.iterator();
```

II. ArrayList

1. 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;
  }
}
```

2. Classe Paire

Classe « Paire »

```
public class Paire<T,U> {
  private final T premier;
  private final U second;

  public Paire(T premier, U second) {
    this.premier = premier;
    this.second = second;
  }

  public T getPremier() {
     return premier;
  }

  public U getSecond() {
    return second;
  }
}
```

3. Classe KeskonrixGestion

1. Reprendre l'attribut trophees de la classe **KeskonrixGestion** afin de transformer le tableau d'objets de la classe « RenseignementTrophee » en une liste d'objets de la classe « Paire ».

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

2. Reprendre la méthode *ajouterTrophee* pour placer une nouvelle paire dans la liste.

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

  public void ajouterTrophee(Gaulois proprietaire, Equipement trophee) {
```

```
}
```

3. Reprendre la méthode *tousLesTrophees* pour récupérer dans chacune des paires de la liste le second élément.

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

  public String tousLesTrophees() {
    String tousLesTrophees = "Tous les trophées du musée sont :\n";
```

```
    return tousLesTrophees;
  }
```

III. Itérateur spécial liste

Keskonrix souhaite aider Panoramix. Il a donc créé la classe « Ingredient » sans difficulté, puis la classe « Potion ».
Comme il ne connaît aucune autre collection, il utilise une ArrayList contenant les ingrédients de la potion magique.

```
public class Ingredient {
   String nom;
   Necessite necessaire;

   public Ingredient(String nom, Necessite necessaire) {
      this.nom = nom;
      this.necessaire = necessaire;
   }

   public Necessite getNecessaire () {
      return necessaire;
   }

   public String toString() {
      return nom;
   }
}

public enum Necessite {
   INDISPENSABLE, AU_CHOIX, OPTIONNEL;
}

public class Potion {
   List<Ingredient> listeIngredients = new ArrayList<>();
   public void ajouterIngredient(Ingredient ingredient) { … }
}
```

Les ingrédients sont ajoutés à l'aide de la méthode *ajouterIngredient(Ingredient ingredient)* qui ajoute les ingrédients grâce à un itérateur. Et comme Keskonrix a bien écouté et qu'il a utilisé une liste il veut utiliser un ListIterator.

Même s'il y a beaucoup plus simple, pouvez-vous l'aider ?

Sur la page suivante, donner le code de la méthode *ajouterIngrédient* en suivant les indications données par les commentaires.

```
public class Potion {
   List<Ingredient> listeIngredients = new ArrayList<>();

  public void ajouterIngredient(Ingredient ingredient) {
   //Si la liste est vide
   //Aide : utiliser les méthodes de l’ArrayList
   if(                                   {

   } else {
                                          {
      Necessite necessaire =
      switch(necessaire) {

      //Si l’ingrédient à ajouter n’est pas indispensable
      case OPTIONNEL:

        break;

      //Si l’ingrédient à ajouter est indispensable
      case INDISPENSABLE:

        break;

      //Si l’ingrédient à ajouter est un ingrédient au choix
      default:  //AU_CHOIX
        boolean ingredientAjoute = false;
        //parcourrir la liste à l’aide d’un ListIterator
        for (



          Ingredient ingredientAcomparer =

          Necessite necessiteAcomparer =

          if(necessiteAcomparer.equals(Necessite.OPTIONNEL)){
            //reculer dans la liste d’un ingrédient

            //ajouter l’ingrédient

            ingredientAjoute = true;
          }
        }
```

```
            if(!ingredientAjoute){
                //ajouter l'ingrédient
```

```
            }
            break;
          }
        }
      }
    }
}
```

IV. Méthode equals

Pour l'instant, dans l'application de Keskonrix, on peut ajouter 2 fois le même ingrédient…

1. Ecrire la méthode *equals* dans la classe **Ingredient** : deux ingrédients sont considérés identiques s'ils ont le même nom.

2. Dans le code de la classe Potion en page 4, modifier la méthode pour n'ajouter l'ingrédient que s'il n'est pas déjà dans la liste.

V. Travail de préparation à l'examen

Pour vous préparer EN AUTONOMIE je vous propose de :
- Relire votre cours,
- De refaire les exercices en téléchargeant le code nécessaire sous Moodle,
- De répondre aux questions ci-dessous (rechercher les réponses dans le cours ou dans le TD).

1. De quelle interface hérite la classe « Collection » ?

2. Cela signifie la classe « Collection » et ses enfants peuvent utiliser l'itérateur « ListIterator » ?

3. Quelles sont les méthodes de la classe « Iterator » ?

4. Quelles sont les méthodes de la classe « ListIterator » ?

5. Je souhaite utiliser une « ArrayList » et je n'ai besoin pour la gérer que des méthodes de la classe « Iterator ». Qu'est-ce que je choisis pour la parcourir un itérateur de type « Iterator » ou de type « ListIterator » ?

6. Une boucle foreach utilise quelles méthodes de l'interface « Iterator » ?

7. Peut-on faire un foreach sur n'importe quel objet héritant de l'interface « Collection » ?

8. Peut-on ajouter un objet dans la liste que l'on est en train de parcourir dans une boucle foreach ?

9. On peut récupérer les éléments d'un énuméré en utilisant la méthode *values*. Par exemple : Necessite[] tableau = Necessite.values() ;
   Comment afficher les éléments de l'énuméré « Necessite » avec une boucle foreach ?

10. De quelles méthodes ont besoin les méthodes *contains*, *containsAll*, *remove*, *removeAll* et *retainAll* pour fonctionner ?

11. Quelle est la signature de la méthode *equals* ?

12. int a = 1 ; int b = 1 ;
    Peut-on écrire a.equals(b) ?

13. Integer a = 1 ; Integer b = 1 ;
    Peut-on écrire a.equals(b) ?

14. Peut-on écrire l'instruction ci-dessous ?
    Necessite.OPTIONNEL.equals(Necessite.OPTIONNEL) ;

15. Quel méthode de la classe « ArrayList » je dois utiliser pour :
    a. savoir si elle est vide ?

    b. savoir si elle contient un élément particulier ?

    c. connaitre sa taille ?

    d. récupérer une chaine contenant tous ses éléments entre crochets ?