Travaux dirigé n°3

« Le parcours des collections, ArrayList,
Itérateur spécial liste, Méthode *equals* »

Cet énoncé vient en appui des diapositives du Cours.

I. Le parcours des collections

Supprimer le deuxième et le 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 une feuille à part, donner le code de la méthode *ajouterIngrédient* en suivant les indications données par les commentaires :

```
public void ajouterIngredient(Ingredient ingredient) {
  //Si la liste est vide
  //Si l'ingrédient à ajouter n'est pas indispensable :
  //Si l'ingrédient à ajouter est indispensable
  //Si l'ingrédient à ajouter est un ingrédient au choix
```

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. Modifier la méthode pour n'ajouter l'ingrédient que s'il n'est pas déjà dans la liste.

```
public void ajouterIngredient(Ingredient ingredient) {
  //Si la liste est vide
(…)
  //Si l'ingrédient à ajouter n'est pas indispensable :
  else {
```

```
    switch (necessaire) {
    (…)
…}
}
```