Correction TD n°9

Java Avancé

—M1 Apprentissage—

Enum, Set

Enum, Enum abstraite, Set

▸ Exercice 1. Calculette

1. Créer une énumération constituée des 4 opérations courantes (addition, soustraction, multiplication, division).
2. On souhaite pouvoir récupérer le symbole associé à l'opération. Pour cela, utiliser un constructeur privé associé à l'énumération.
3. On souhaite pouvoir afficher ce symbole. Par exemple, `System.out.println(Operation.PLUS);` affichera +. Indice : une énumération est aussi un objet.
4. On souhaite maintenant effectuer les calculs liés à ces opérations. Écrire une méthode `compute` prenant deux doubles et retournant le résultat de l'opération. Réfléchir à l'intérêt d'une énumération abstraite ici plutôt qu'une méthode utilisant un switch.
5. Tester avec ces tests JUnit : OperationTest

✂ -

si switch, risqué lors de l'ajout d'une opération (oubli, exception à runtime etc).

```java
public enum Operation {
  PLUS('+') {
    @Override
    public double compute(double a, double b) {
      return a+b;
    }
  }, MINUS('-') {
    @Override
    public double compute(double a, double b) {
      return a-b;
    }
  };

  private Operation(char c) {
    this.symbol = c;
  }

  @Override
  public String toString() {
    return Character.toString(symbol);
```

```
  }

  private final char symbol;
  abstract public double compute(double a, double b);


  public static void main(String[] args) {
    System.out.println(Operation.PLUS); //affiche +
    double a = 7.0;
    double b = 1.6;
    Operation o = Operation.PLUS;
    System.out.printf("%f %s %f = %f\n", a, o, b, o.compute(a,b));
  }
}
```

▸ Exercice 2. Swag

On veut définir une méthode `String swag(String txt, int style);` dans une classe `Swag` qui renvoie le texte txt avec un certain style passé également en argument (on n'utilise pas d'énumération pour l'instant). L'entier style indique l'effet à appliquer. Par exemple, pour le texte "miage" :

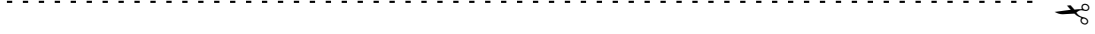
- — CROSS affiche m+i+a+g+e
- — KIKOO affiche MiAgE
- — CROSS|KIKOO affiche M+i+A+g+E

1. Fixer CROSS et KIKOO comme des constantes entières et écrire la méthode swag. Des combinaisons sont possibles, par exemple `swag(‘‘miage’’,CROSS|KIKOO)` doit fonctionner.
2. Tester avec SwagTest1
3. Utiliser une énumération (qu'on appelle `STYLE`) plutôt que des constantes. Pour passer plusieurs éléments, vous pouvez utiliser un `Set`. Modifier votre méthode swag pour qu'elle prenne un `Set` en second argument.
4. Faire appel à votre méthode dans un main. Pour créer facilement un Set, utiliser `EnumSet.of()`.
5. Modifier votre énumération pour qu'elle contienne une méthode abstraite `applyStyle` appliquant la transformation associée. Modifier votre méthode swag pour qu'elle l'utilise.
6. Si vous avez placé CROSS avant KIKOO dans votre énumération, que remarquez-vous sur ce qui est renvoyé si les 2 styles sont demandés ? Pour remédier à cela, utilisez un Set qui préserve l'ordre d'insertion, comme `LinkedHashSet`.
7. Tester avec ces tests JUnit : SwagTest2

```java
private final static int CROSS = 1;
  private final static int KIKOU = 2;

  static String swag(String txt, int flags) {
    StringBuilder myNewString=new StringBuilder() ;
    switch (flags) {
    case CROSS:
      for (int i = 0 ; i < txt.length() ; i++) {
        if(i<txt.length()-1)
        {
          myNewString.append(txt.charAt(i)).append("+") ;

        }  else {
          myNewString.append(txt.charAt(i)) ;
        }
      }
      break;
    case KIKOU:
      for (int i = 0 ; i < txt.length() ; i++) {
        if(i%2==0)
        {
          myNewString.append(Character.toUpperCase(txt.charAt(i))) ;

        }  else {
          myNewString.append(txt.charAt(i)) ;
        }
      }
      break;
    case KIKOU|CROSS:
      for (int i = 0 ; i < txt.length() ; i++) {
        if(i%2==0)
        {
          myNewString.append(Character.toUpperCase(txt.charAt(i))) ;

        }  else {
          myNewString.append(txt.charAt(i)) ;
        }
        if(i<txt.length()-1)
        {
          myNewString.append("+") ;

        }  else {

        }
      }
      break;
    default:
      throw new IllegalArgumentException();

    }
    return myNewString.toString();
  }

  public static void main(String[] args) {
    System.out.println(CROSS|KIKOU); //renvoie 3 : ne pas mettre 0 et 1 !
    System.out.println(swag("miage", CROSS));
    System.out.println(swag("miage", KIKOU));
    System.out.println(swag("miage", CROSS|KIKOU));
  }
}
```

```java
public class Swag {

  enum STYLE {
    CROSS {
```

```
5       @Override
6       String applyStyle(String txt) {
7         StringBuilder myNewString = new StringBuilder();
8         for (int i = 0 ; i < txt.length() ; i++) {
9           if(i<txt.length()-1){
10            myNewString.append(txt.charAt(i)).append("+") ;
11
12          } else {
13            myNewString.append(txt.charAt(i)) ;
14          }
15        }
16        return myNewString.toString();
17      }
18    }, KIKOU {
19      @Override
20      String applyStyle(String txt) {
21        StringBuilder myNewString = new StringBuilder();
22        for (int i = 0 ; i < txt.length() ; i++) {
23          if(i%2==0) {
24            myNewString.append(Character.toUpperCase(txt.charAt(i))) ;
25
26          } else {
27            myNewString.append(txt.charAt(i)) ;
28          }
29        }
30        return myNewString.toString();
31      }
32    };
33
34    abstract String applyStyle(String txt);
35  }
36
37  static String swagEnum(String txt, Set<STYLE> styles) {
38    String myNewString=txt;
39
40    for(STYLE s:styles) {
41      myNewString = s.applyStyle(myNewString);
42    }
43
44    return myNewString;
45  }
46
47
48  public static void main(String[] args) {
49    System.out.println(CROSS|KIKOU); //renvoie 3 : ne pas mettre 0 et 1 !
50    System.out.println(swag("miage", CROSS));
51    System.out.println(swag("miage", KIKOU));
52    System.out.println(swag("miage", CROSS|KIKOU));
53
54
55    LinkedHashSet<STYLE> hs = new LinkedHashSet<Swag.STYLE>();
56    hs.add(STYLE.KIKOU);
57    hs.add(STYLE.CROSS);
58
59    System.out.println(swagEnum("miage", EnumSet.of(STYLE.CROSS)));
60    System.out.println(swagEnum("miage", EnumSet.of(STYLE.KIKOU)));
61    //affiche mal, car fait d'abord le cross puis le kikou, mais pas d'effet kikou car une lettre sur 2 !
62    System.out.println(swagEnum("miage", EnumSet.of(STYLE.CROSS,STYLE.KIKOU)));
63    System.out.println(swagEnum("miage", hs));
64  }
65 }
```



▸ **Exercice 3. Sac**

Une structure données stockant plusieurs fois le même élément (au sens du equals) et le nombre de fois où cet élément apparait est appelé Bag ou MultiSet. Elle n'existe pas dans la JDK, mais est souvent utile.

1. Écrire une classe (paramétrée) pour une telle structure de données, avec les méthodes d'ajout, suppression (d'une occurrence) et de comptage (nombre d'occurrences d'un objet). Ces opérations doivent s'effectuer avec la meilleure complexité possible.
2. Ajouter une méthode `Iterator<Map.Entry<T,Integer>> iterator()` qui renvoie un itérateur sur les couples stockés dans votre structure. La solution s'effectue en une seule ligne !
3. Ajouter une énumération permettant de choisir l'ordre pour l'itération (au moment de la construction du Bag (vous garderez un constructeur sans argument qui utilise aucun ordre spécifique). Les choix possibles doivent être : ordre d'insertion, ordre naturel (au sens compareTo) et aucun ordre spécifique. Est-ce que cela impose des contraintes sur le type d'objets stockés ? (Normalement, il aurait fallu faire une interface commune avec différentes classes concrètes selon le type, mais le but est de vous faire utiliser des énumérations !).
4. Utiliser une méthode abstraite dans votre énumération pour simplifier le code du constructeur.
5. Tester avec ces tests JUnit : BagTest1
6. On veut pouvoir exécuter le code suivant :

```
Bag<String> bag=new Bag<>(BagOrder.ANY);
bag.add("toto");
bag.add("toto");
bag.add("titi");
for(Map.Entry<String,Integer> entry:bag)
   System.out.println(entry.getKey()+'='+entry.getValue());
```

   . Que faut-il faire pour que cela fonctionne ?
7. Réfléchir à quelle interface de Collection votre implémentation pourrait coller. Vous avez le droit d'utiliser `AbstractCollection`. Est-ce que cela pose un problème pour l'itérateur ? Tester avec le code suivant :

```
Collection<String> bag = new Bag<>(BagOrder.INSERT);
bag.add("denis");
bag.add("cornaz");
bag.add("denis");
for(String s:bag) {
  System.out.println(s);
  //affiche denis denis cornaz
}
```

8. Tester avec ces tests JUnit : BagTest2

✂ -

```
1  public class Bag<T> implements Iterable<Map.Entry<T, Integer>>{
2
3    enum BagOrder {
4      ANY {
5        @Override
6        <T> Map<T, Integer> createMap() {
7          return new HashMap<>();
8        }
9      },INSERT {
10       @Override
11       <T> Map<T, Integer> createMap() {
12         return new LinkedHashMap<>();
13       }
14     },NATURAL {
15       @Override
16       <T> Map<T, Integer> createMap() {
17         return new TreeMap<>();
18       }
19     };
20
21     abstract <T> Map<T,Integer> createMap();
22   }
23
24   private final Map<T,Integer> map;
25
26
27   public Bag(BagOrder order) {
28     map = order.createMap();
29   }
30
31   public void add(T t) {
32     if(!map.containsKey(t)) {
33       map.put(t, 1);
34     }
35     else {
36       map.put(t, map.get(t)+1);
37     }
38   }
39
40   public boolean remove(T o) {
41     Integer nbOccOfo = map.get(o);
42     if (nbOccOfo == null) {
43       return false;
44     }
45     if (nbOccOfo == 1) {
46       map.remove(o);
47     } else {
48       map.put(o, nbOccOfo - 1);
49     }
50     return true;
51   }
52
53   public int count(T t) {
54     return map.get(t);
55   }
56
57   @Override
58   public Iterator<Entry<T, Integer>> iterator() {
59     return map.entrySet().iterator();
60   }
61
62   public static void main(String[] args) {
63     /*
64      Bag<String> bag=new Bag<>(BagOrder.ANY);
65      bag.add("toto");
66      bag.add("toto");
67      bag.add("titi");
```

```
    for(Map.Entry<String,Integer> entry:bag)
      System.out.println(entry.getKey()+'='+entry.getValue());
    */
    Bag<?> bag=new Bag<String>(BagOrder.ANY);
    //Iterator<Map.Entry<?,Integer>> it=bag.iterator();
    Iterator< ? extends Map.Entry< ?,Integer>> it=bag.iterator() ;
  }
}
```

```
public class Bag<T> extends AbstractCollection<T> implements Collection<T>{

  enum BagOrder {
    ANY {
      @Override
      <T> Map<T, Integer> createMap() {
        return new HashMap<>();
      }
    },INSERT {
      @Override
      <T> Map<T, Integer> createMap() {
        return new LinkedHashMap<>();
      }
    },NATURAL {
      @Override
      <T> Map<T, Integer> createMap() {
        return new TreeMap<>();
      }
    };

    abstract <T> Map<T,Integer> createMap();
  }

  private final Map<T,Integer> map;


  public Bag(BagOrder order) {
    map = order.createMap();
  }

  @Override
  public boolean add(T t) {
    if(!map.containsKey(t)) {
      map.put(t, 1);
    }
    else {
      map.put(t, map.get(t)+1);
    }
    return true;
  }

  @SuppressWarnings("unchecked")
  public boolean remove(Object o) {
    // Why Object instead of E ? when trying to remove an object which is
    // not in the collection, whatever the type of this object is, the
    // remove method should return false
    Integer nbOccOfo = map.get(o);
    if (nbOccOfo == null) {
      return false;
    }
    // Suppressing the type safety warning can be done since we can ensure
    // that
    // at that point o is of E type ! (occurs in map)
    if (nbOccOfo == 1) {
      map.remove(o);
    } else {
      map.put((T) o, nbOccOfo - 1);
    }
```

```
59
60    return true;
61  }
62
63  public int count(T t) {
64    return map.get(t);
65  }
66
67  @Override
68  //public Iterator<Entry<T, Integer>> iterator() {
69  //on itere plus sur des couples, mais sur des T
70  public Iterator<T> iterator() {
71    return new Iterator<T>() {
72      //doit prendre en compte le nb d'occ
73
74      private final Set<Map.Entry<T, Integer>> myEntrySet = map.entrySet();
75
76      private final Iterator<Map.Entry<T, Integer>> myIterator = myEntrySet
77      .iterator();
78
79      private int nbOccurencesToDo = 0;
80
81      private T actualKey = null;
82
83      // Read access to enclosing field CollectionEnumMapBag<E>.myMap is
84      // emulated by a synthetic accessor method. Increasing its
85      // visibility will
86      // improve your performance
87      public boolean hasNext() {
88        // we have to consider the special case of the last key which
89        // may have
90        // several occurences
91        return (nbOccurencesToDo > 0) || myIterator.hasNext();
92      }
93
94      public T next() {
95        if (!hasNext()) {
96          return null;
97        }
98        if (nbOccurencesToDo == 0) {
99          // Case of a new Key
100         Map.Entry<T, Integer> myEntry = myIterator.next();
101         nbOccurencesToDo = myEntry.getValue();
102         actualKey = myEntry.getKey();
103       }
104       nbOccurencesToDo--;
105       return actualKey;
106     }
107   };
108 }
109
110 @Override
111 public int size() {
112   return map.size(); //a voir si c'est pas la somme des occs
113 }
```