IUT de Paris Descartes – Base de la Programmation Objet

Travaux Pratiques – Sujet n°6

Objectifs

– Utiliser les collections de la bibliothèque standard

Un peu de poésie

La classe `Poète` dont le code est fourni, permet de générer aléatoirement des textes. Pour donner une certaine cohérence aux textes qu'il produit, un 'poète' se base sur des textes existants qui lui auront été préalablement soumis.

Un programme de test (`Appli.java`), la classe `Poète` (`Poète.java`) ainsi que de nombreux textes (`lafontaine.txt` et tout le répertoire `textes`) que nous soumettrons à notre poète se trouvent sur le serveur.

Le principe suivi par notre poète pour produire un texte est le suivant :

1. Il commence par produire les deux premiers mots d'un des textes qu'il connaît.
2. À partir de ces deux premiers mots, il en choisi un troisième parmi tous les mots qu'il a vu suivre ces deux-là dans les textes qui lui ont été soumis.
3. Il recommence le même procédé avec les deux derniers mots qu'il a produits.
4. Toutefois, si les deux derniers mots produits terminent un texte existant, il peut choisir de terminer ici sa production.

Pour illustrer cet algorithme, considérons le texte suivant :

"Montrez-moi vos algorithmes et dissimulez vos structures de données, et je serais mystifié. Montrez-moi vos structures de données, et vos algorithmes seront prévisibles."

En faisant l'hypothèse que notre poète ne connaisse que ce texte, il doit être capable de produire :

"Montrez-moi vos algorithmes seront prévisibles."

En effet, comme notre poète ne connaît qu'un seul texte et que celui-ci commence par les deux mots *'Montrez-moi vos'*, ces derniers doivent absolument débuter le texte produit (règle 1). Dans le texte qui lui a été soumis, ces deux mots sont suivis par *'algorithmes'* et par *'structures'*. Ici, le poète doit faire un choix (règle 2) et dans le texte résultant, nous pouvons voir qu'il a choisi *'algorithmes'*. Les deux derniers mots produits sont maintenant *'vos algorithmes'* et la règle 3 impose de choisir le mot suivant parmi ceux qui suivent *'vos algorithmes'* dans le texte original. Là encore, deux possibilités s'offrent au poète. Il peut choisir parmi *'et'* et *'seront'*. Une fois le choix de *'seront'* effectué, le poète n'a plus de choix en possible. En effet, seul *'prévisibles.'* suit les deux mots *'algorithmes seront'* dans le texte original. Et comme *'seront prévisibles.'* termine le texte, le poète peut décider de cesser sa production ici (en fait c'est le seul choix possible car *'seront prévisibles.'* n'apparaît nulle part ailleurs dans le texte original).

Compréhension du programme fourni

1. Parcourez l'ensemble du code en notant que :
   - Le programme principal déclare un poète `p`, lui soumet différents textes par un appel à la méthode statique `leçon`, affiche les statistiques sur les connaissances du poète et affiche un texte inventé par le poète.
   - La méthode `leçon` trouve dans un fichier dont le nom est passé en paramètre, les noms des fichiers contenant les textes à soumettre au poète. Pour chacun de ces fichiers, la méthode `Poète.apprendre` est invoquée.
   - La méthode `Poète.apprendre` lit les mots du texte sans se préoccuper des signes de ponctuation. Toute suite de caractères ne contenant pas d'espace est considérée comme étant un mot.

Connaissance du poète

Il s'agit à présent de compléter la partie privée de la classe `Poète`. L'algorithme présenté ci-dessus nous impose de connaître :
- toutes les paires de mots débutant les textes qui ont été soumis,
- pour chaque paire de mots, la liste des mots pouvant être mis à sa suite.

2. Au sein de la classe `Poète`, définissez une classe statique et privée nommée `Préfixe`. Cette classe doit permettre de stocker une paire de mots. Elle doit disposer d'un constructeur et il doit être possible de dupliquer un préfixe. Cette classe est utilitaire et ses attributs pourront être déclarés publics.

Solution:

```
public class Poète {
  private static class Préfixe {
    public String avantDernier, dernier;

    public Préfixe(String avantDernier, String dernier) {
      this.avantDernier = avantDernier;
      this.dernier = dernier;
    }

    public Préfixe(Préfixe p) {
      this(p.avantDernier, p.dernier);
    }
  }
  ...
}
```

3. Définissez le type de l'attribut `débuts` de la classe `Poète`. Il doit stocker toutes les paires de mots débutant les textes. Choisissez pour cela un des types de la bibliothèque standard permettant de stocker une séquence d'éléments. Les opérations que nous aurons à faire sont l'insertion d'un nouvel élément et la sélection aléatoire d'un élément parmi ceux présents.

Solution:

```
public class Poète {
  ...
  private ArrayList<Préfixe> débuts;
```

```
  ...
}
```

4. Définissez le type de l'attribut `savoir` de la classe `Poète`. Il doit associer à chaque préfixe rencontré dans un texte la séquence de mots contenant tous les mots pouvant être mis à la suite du préfixe. Employez la classe `HashMap` à cette fin. Les seules opérations requises sur la séquence de mots associée à un préfixe sont l'insertion et la sélection aléatoire.

Solution:

```
public class Poète {
  ...
  private HashMap<Préfixe, ArrayList<String>> savoir;
  ...
}
```

5. Pour qu'un préfixe puisse être employé en tant que clé d'une table de hachage, la classe `Préfixe` doit disposer d'une méthode de hachage et d'une méthode de comparaison. Leurs prototypes respectifs sont `int hashCode()` et `boolean equals(Object obj)`. Ajoutez ces deux méthodes.

Solution:

```
public class Poète {
  private static class Préfixe {
    ...
    public int hashCode() {
      return avantDernier.hashCode() + dernier.hashCode();
    }

    public boolean equals(Object obj) {
      if (obj == null || obj.getClass() != this.getClass())
        return false;
      if (obj == this)
        return true;
      Préfixe p = (Préfixe)obj;
      return p.avantDernier.equals(avantDernier) &&
                                  p.dernier.equals(dernier);
    }
  }
  ...
}
```

Actions du poète

Il s'agit à présent de compléter le code des méthodes de la classe `Poète`. Pour toutes les questions qui suivent, vous devez vous reporter à la documentation de la classe `HashMap`.

6. Complétez le constructeur de manière à ce que tout poète soit initialement complètement ignorant.

Solution:

```
public class Poète {
  ...
  public Poète() {
    débuts = new ArrayList<Préfixe>();
    savoir = new HashMap<Préfixe, ArrayList<String>>();
  }
  ...
}
```

7. Complétez la méthode `neSaitRien` qui détermine si son savoir est vide ou pas.

Solution:

```
public class Poète {
  ...
  public boolean neSaitRien() {
    return savoir.isEmpty();
  }
  ...
}
```

8. Complétez la méthode `printStatistiques` en suivant les commentaires présents dans le source.

Solution:

```
public class Poète {
  ...
  public void printStatistiques() {
    int nbDébuts = débuts.size();
    int nbPréfixes = savoir.size();
    int max = 0, nbSup = 0, cumul = 0;
    for (ArrayList<String> mots : savoir.values()) {
      int nb = mots.size();
      if (nb > max)
        max = nb;
      if (nb > 1)
        ++nbSup;
      cumul += nb;
    }
    System.out.println("nbre de textes : " + nbDébuts);
    System.out.println("nbre de préfixes : " + nbPréfixes);
    System.out.println(
      "nbre maximal de mots liés à un même préfixe : " + max);
    System.out.println(
      "nbre de préfixes associés à au - 2 mots : "+ nbSup);
    if (nbPréfixes != 0)
      System.out.println(
          "nbre moyen de mots associés aux préfixes : " +
                                ((float) cumul / nbPréfixes));
  }
  ...
}
```

9. Complétez la méthode `apprendre` en suivant les commentaires présents dans le source.

Solution:

```
public class Poète {
  ...
  public void apprendre(String nom) throws
      FileNotFoundException, NoSuchElementException {
    Scanner sc = new Scanner(new File(nom));
    // si le fichier contient - de 2 mots, les deux lectures
    // suivantes lèvent l'exception NoSuchElementException
    Préfixe p = new Préfixe(sc.next(), sc.next());
    débuts.add(new Préfixe(p));
    while (sc.hasNext()) {
      String s = sc.next();
      ajouterMotAPréfixe(p, s);
      p.avantDernier = p.dernier;
      p.dernier = s;
    }
    ajouterMotAPréfixe(p, "");
    sc.close();
  }

  private void ajouterMotAPréfixe(Préfixe p, String s) {
    if (savoir.containsKey(p))
      savoir.get(p).add(s);
    else
      savoir.put(new Préfixe(p),
                 new ArrayList<String>(Arrays.asList(s)));
  }
  ...
}
```

10. Complétez la méthode `inventer` en suivant les commentaires présents dans le source.

Solution:

```
public class Poète {
  ...
  public String inventer() {
    assert (!neSaitRien());
    StringBuilder sb = new StringBuilder();
    Random rd = new Random();
    Préfixe p = new Préfixe(
                  débuts.get(rd.nextInt(débuts.size()))
                );
    sb.append(p.avantDernier + "␣" + p.dernier);
    while (savoir.containsKey(p)) {
      ArrayList<String> mots = savoir.get(p);
      String s = mots.get(rd.nextInt(mots.size()));
      sb.append("␣" + s);
      p.avantDernier = p.dernier;
      p.dernier = s;
    }
```

```
    return sb.toString();
  }
  ...
}
```