```
public class Couple<TCle , TVal> {

  private TCle cle;
  private TVal val;

  public Couple(TCle c, TVal v) {
    cle = c;
    val = v;
  }

  public void setCle(TCle nouvelleCle) {
    cle = nouvelleCle;
  }

  public void setVal(TVal nouvelleVal) {
    val = nouvelleVal;
  }

  public TCle getCle() {
    return cle;
  }

  public TVal getVal() {
    return val;
  }

}
```

```
/**
 * TableDeCorrespondance.java : interface du TA Table de correspondance
 *
 * Created: Thu Nov  4 15:10:15 2004
 *
 * @author <a href="mailto:Jean−Christophe.Engel@irisa.fr">Jean−Christophe Engel
</a>
 * @version 1
 */

public interface TableDeCorrespondance<TCle , TVal> {
    /**
     * initialise une table vide
     * @param taille : taille maximale de la table
     */
  //     public TableDeCorrespondance(int taille);

    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
    // méthodes publiques de parcours
    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
    /**
     * debut : initialise le parcours
     */
    public void debut();

    /**
     * enFin   : vrai si le parcours est terminé
     * @return vrai si le parcours est terminé
     */
    public boolean enFin();

    /**
     * cleCourante : donne la clé de l'association courante
     * @pre non enFin()
     * @return clé de l'association courante
     */
    public TCle cleCourante();

    /**
     * valeurCourante : donne la valeur de l'association courante
     * @pre non enFin()
     * @return valeur de l'association courante
     */
    public TVal valeurCourante();

    /**
     * avancer : passe à l'élément suivant
     * @pre non enFin()
     */
    public void avancer();
    //
```



```
    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
    // méthodes publiques d'accès
    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
    /**
     * Présent : détermine si une clé est présente
     * @param cle : clé cherchée
     * @return vrai si la clé est présente
     */
    public boolean present(TCle cle);

    /**
     * Valeur  : valeur associée à une clé donnée
     * pré     : la clé est présente
     * @param cle : clé dont on veut la valeur associée
     * @return valeur de l'association dont on donne la clé
     */
    public TVal valeur(TCle cle);

    /**
     * Ajouter : ajoute une association (clé, valeur)
     * laisse la table inchangée si la clé est présente ou si la table est plein
e
     * @param cle : clé à ajouter
     * @param val : valeur associée à la clé
     */
    public void ajouter(TCle cle, TVal val);

    /**
     * Modifier : remplace la valeur associée à une clé
     * laisse la table inchangée si la clé est absente
     * @param cle : clé dont on veut modifier la valeur
     * @param val : nouvelle valeur associée à la clé
     */
    public void modifier(TCle cle, TVal val);

    /**
     * Retirer : retire l'association dont la clé est donnée
     *  laisse la table inchangée si la clé est absente
     * @param cle : clé de l'association à supprimer
     */
    public void retirer(TCle cle);

}
```

```
public class TableDeCorrespondanceTabloTrie<TCle extends Comparable<TCle>, TVal>
  implements TableDeCorrespondance<TCle, TVal> {

  private static final int TAILLE_MAX = 50;
  private Object[] table;
  private int nbElements;
  private int position;

  public TableDeCorrespondanceTabloTrie(int taille) {
    table = new Object[taille];
    nbElements = 0;
    position = 0;
  }

  public void debut() {
    position = 0;
  }

  public boolean enFin() {
    return position == nbElements;
  }

  private boolean estALAvantDernier() {
    return position == nbElements − 1;
  }

  public boolean auDebut() {
    return position == 0;
  }

  public TCle cleCourante() {
    if (enFin())
      throw new RuntimeException("Fin de la table");
    return ((Couple<TCle,TVal>) table[position]).getCle();
  }

  public TVal valeurCourante(){
    if (enFin())
      throw new RuntimeException("Fin de la table");
    return ((Couple<TCle,TVal>) table[position]).getVal();
  }

  public void avancer() {
    if (enFin())
      throw new RuntimeException("Fin de la table");
    position++;
  }

  public void reculer() {
    if (auDebut())
      throw new RuntimeException("Debut de la table");
    position−−;
  }

  public boolean present(TCle cle) {
    int a;

    debut();
    while(!enFin()) {
      a = cleCourante().compareTo(cle);
      if (a == 0)
        return true;
      if (a > 0)
        return false;
      avancer();
    }
    return false;
  }

  public TVal valeur(TCle cle) {
    if(present(cle))
      return valeurCourante();
    else
      throw new RuntimeException("Cle absente");
  }

    public void ajouter(TCle cle, TVal val) {
      int i, place;

      if(nbElements == table.length)
        return;

      debut();

      while(!enFin() && cleCourante().compareTo(cle) < 0)
        avancer();
      if(!enFin() && cleCourante().compareTo(cle) == 0)
        return;

      place = position;

      while(!enFin())
        avancer();

      while(position != place) {
        reculer();
        table[position + 1] = table[position];
      }

      table[place] = new Couple<TCle, TVal>(cle, val);
      nbElements++;
    }

  public void modifier(TCle cle, TVal val) {
    if(present(cle))
      ((Couple<TCle, TVal>) table[position]).setVal(val);
  }

  public void retirer(TCle cle) {
    if(present(cle)) {
      while (!estALAvantDernier()) {
        table[position] = table[position + 1];
        avancer();
      }
      table[position] = null;
      nbElements−−;
    }
  }

}
```

```
import java.util.*;

public class Test{

  public static void afficher(TableDeCorrespondance t) {
    t.debut();
    while(!t.enFin()) {
      System.out.println(t.cleCourante() + ":" +
                         t.valeurCourante());
      t.avancer();
    }
  }

  public static void main(String[] args) {
     TableDeCorrespondanceVect<String, Integer> t1 =
      new TableDeCorrespondanceVect<String, Integer>(5);
    t1.ajouter("Albert", 1);
    t1.ajouter("Guy", 4);
    t1.ajouter("Fred", 3);
    t1.ajouter("Bernard", 2);
    t1.ajouter("Tristan", 5);
    afficher(t1);
    System.out.println("--------------");
    t1.modifier("Bernard", 1981);
     afficher(t1);
    System.out.println("--------------");
    t1.retirer("Fred");
    afficher(t1);
    System.out.println("--------------");
    t1.retirer("Albert");
    afficher(t1);
    System.out.println("--------------");
    t1.retirer("Tristan");
    afficher(t1);
    System.out.println("--------------");

  }
}
```