Programmation avancée en Java
Travaux dirigés – Corrigés

Par Dr. Samia GAMOURA-CHEHBI

Maj 2016

INFORMATIQUE

Le TD n°8 reprend tous les autres TD du cours JAVA2. Ainsi, sont présentés ci-dessous, tous les codes sources correspondant à l'application construite dans le TD n°8.

```
package commande;

/**
 * Titre :        Formation Java Pratique
 * Description :  Formation Java Pratique - Exercices
 * Copyright :
 * Société :      DR. S. GAMOURA
 * @author JRE
 * @version 1.0
 */


/**
 * Classe représentant un caddy d'articles
 */
public class Caddy {

  /** Login de la personne utilisant le caddy */
  private String _login;
  /** Commande du caddy */
  private Commande _commande;

  /**
   * Constructeur
   */
  public Caddy() {
  }


      // GETTER / SETTER
  /**
   * Affectation de la commande
   * @param commande la Commande
   */
  public void setCommande(Commande commande) {
    _commande = commande;
  }

  /**
   * Récupération de la commande
   * @return la Commande
   */
  public Commande getCommande() {
    return _commande;
  }

  /**
   * Affectation du login de l'utilisateur
   * @param login Le login de l'utilisateur
   */
  public void setLogin(String login) {
```

```
    _login = login;
  }

  /**
   * Récupération du login de l'utilisateur
   * @return Le login de l'utilisateur
   */
  public String getLogin()
  {
    return _login;
  }


      // METHODES
  /**
   * Ajout d'une ligne de commande à partir de ses attributs
   * @param nomProduit le nom du produit à insérer dans la
commande
   * @param quantity la quantité de produit désirée
   */
  public void add(String nomProduit, int quantity) {
        _commande.add(nomProduit, quantity);
  }

}
```

```
package commande;

/**
 * Titre :          Formation Java Pratique
 * Description :    Formation Java Pratique - Exercices
 * Copyright :
 * Société :        DR. S. GAMOURA
 * @author JRE
 * @version 2.0
 */

import java.util.*;
import java.text.*;

/**
 * classe représentant une commande
 */
public class Commande {
  //------------------------------------------------------------
---------------------
  // Members
  //------------------------------------------------------------
---------------------
  /** Référence de la commande */
  private String _ref;
  /** Date de la commande */
  private Date _date;
  /** Nom du Client */
  private String _customer;
  /** Lignes de commande */
  private Vector _lignesCommande;
```

```
  //-----------------------------------------------------------
---------------------
  // Constructors
  //-----------------------------------------------------------
---------------------
  /**
   * Création d'une commande
   * @param ref la référence de la commande
   * @param customer le client
   * @return une instance de Commande
   */
  public Commande(String ref, String customer) {
    _ref = ref;
    _customer = customer;
    _lignesCommande = new Vector();
    _date = new Date();
  }

  //-----------------------------------------------------------
---------------------
  // Acessors/Modifiers
  //-----------------------------------------------------------
---------------------
  /**
   * Retourne la référence de la commande
   * @return la référence
   */
  public void setCustomer(String customer) {
    _customer = customer;
  }

  /**
   * Retourne le nom du client
   * @return le nom du client
   */
  public String getCustomer() {
    return _customer;
  }

  /**
   * Retourne la référence de la commande
   * @return la référence
   */
  public String getRef() {
    return _ref;
  }

  /**
   * Retourne la date de la commande
   * @return la date de la commande
   */
  public String getDate() {
    return
DateFormat.getDateInstance(DateFormat.SHORT).format(_date);
  }

  /**
   * Retourne les différentes lignes de la commandes
   * @return une liste de LigneCommande
   */
  public Vector getLignesCommande() {
```

```
    return _lignesCommande;
  }

  //------------------------------------------------------------
---------------------
  // Methods
  //------------------------------------------------------------
---------------------
  /**
   * Ajouter une ligne de commande
   * @param lg une ligne de commande
   */
  public void add(LigneCommande lg) {
    _lignesCommande.addElement(lg);
  }
  /**
   * Ajouter une ligne de commande à partir de ses attributs
   * @param nomProduit le nom du produit à insérer dans la
commande
   * @param quantity la quantité de produit désirée
   */
  public void add(String nomProduit, int quantity) {
    LigneCommande lg = new LigneCommande(nomProduit,
quantity);
    this.add(lg);
  }

}
```

```
package commande;

/**
 * Titre :         Formation Java Pratique
 * Description :   Formation Java Pratique - Exercices
 * Copyright :
 * Société :       DR. S. GAMOURA
 * @author JRE
 * @version 2.0
 */

import java.util.*;
import java.text.*;

/**
 * classe représentant une commande
 */
public class Commande {
  //------------------------------------------------------------
---------------------
  // Members
  //------------------------------------------------------------
---------------------
  /** Référence de la commande */
  private String _ref;
  /** Date de la commande */
  private Date _date;
  /** Nom du Client */
  private String _customer;
  /** Lignes de commande */
```

```
  private Vector _lignesCommande;

  //-------------------------------------------------------------
---------------------
  // Constructors
  //-------------------------------------------------------------
---------------------
  /**
   * Création d'une commande
   * @param ref la référence de la commande
   * @param customer le client
   * @return une instance de Commande
   */
  public Commande(String ref, String customer) {
    _ref = ref;
    _customer = customer;
    _lignesCommande = new Vector();
    _date = new Date();
  }

  //-------------------------------------------------------------
---------------------
  // Acessors/Modifiers
  //-------------------------------------------------------------
---------------------
  /**
   * Retourne la référence de la commande
   * @return la référence
   */
  public void setCustomer(String customer) {
    _customer = customer;
  }

  /**
   * Retourne le nom du client
   * @return le nom du client
   */
  public String getCustomer() {
    return _customer;
  }

  /**
   * Retourne la référence de la commande
   * @return la référence
   */
  public String getRef() {
    return _ref;
  }

  /**
   * Retourne la date de la commande
   * @return la date de la commande
   */
  public String getDate() {
    return
DateFormat.getDateInstance(DateFormat.SHORT).format(_date);
  }

  /**
   * Retourne les différentes lignes de la commandes
   * @return une liste de LigneCommande
```

```
   */
  public Vector getLignesCommande() {
    return _lignesCommande;
  }

  //------------------------------------------------------------
---------------------
  // Methods
  //------------------------------------------------------------
---------------------
  /**
   * Ajouter une ligne de commande
   * @param lg une ligne de commande
   */
  public void add(LigneCommande lg) {
    _lignesCommande.addElement(lg);
  }
  /**
   * Ajouter une ligne de commande à partir de ses attributs
   * @param nomProduit le nom du produit à insérer dans la
commande
   * @param quantity la quantité de produit désirée
   */
  public void add(String nomProduit, int quantity) {
    LigneCommande lg = new LigneCommande(nomProduit,
quantity);
    this.add(lg);
  }

}
```

```
package commande;

/**
 * Titre :        Formation Java Pratique
 * Description :  Formation Java Pratique - Exercices
 * Copyright :
 * Société :      DR. S. GAMOURA
 * @author JRE
 * @version 2.0
 */

/** classe représentant une ligne de commande
 */
public class LigneCommande {
  //------------------------------------------------------------
---------------------
  // Members
  //------------------------------------------------------------
---------------------
  /** Libellé du produit commandé */
  private String _longName;
  /** Quantité commandée */
  private int _quantity;

  //------------------------------------------------------------
---------------------
  // Constructors
```

```
  //--------------------------------------------------------------
---------------------
  /**
   * Création d'une ligne de commande
   * @param name le nom de la bouteille commandée
   * @param qte la quantité commandée
   * @return une instance de LigneCommande
   */
  public LigneCommande(String name, int qte) {
    _quantity = qte;
    _longName = name;
  }

  //--------------------------------------------------------------
---------------------
  // Acesseurs / Modifiers
  //--------------------------------------------------------------
---------------------
  /**
   * Retourne le nom de la bouteille commandée
   * @return le libellé de la bouteille de vin
   */
  public String getName() {
    return _longName;
  }

  /**
   * Retourne la quantité commandée
   * @return la quantité commandée
   */
  public int getQuantity() {
    return _quantity;
  }
}
```

```
package production;

/**
 * Titre :          Formation Java Pratique
 * Description :    Formation Java Pratique - Exercices
 * Copyright :      Dr. S. GAMOURA 2001
 * Société :        DR. S. GAMOURA
 * @author ADO
 * @version 1.0
 */

public interface Comparable {
  public abstract boolean estPlusCher(Object o);
}
```

```
package production;

/**
 * Titre :          Formation Java Pratique
 * Description :    Formation Java Pratique - Exercices
 * Copyright :
 * Société :        DR. S. GAMOURA
 * @author JRE
```

```
 * @version 2.0
 */

/**
 * Classe ABSTRAITE représentant un Produit générique
 */
public abstract class Produit {
  //-------------------------------------------------------------
---------------------
  // Members
  //-------------------------------------------------------------
---------------------
  private Integer _ref;

  //-------------------------------------------------------------
---------------------
  // Acessors/Modifiers
  //-------------------------------------------------------------
---------------------

  /**
   * Retourne la référence du produit
   * @return _ref la référence du produit
   */
  public Integer getRef() {
    return _ref;
  }
  /**
   * Met à jour la référence du produit
   * @param ref la référence du produit
   */
  public void setRef(Integer ref) {
    _ref = ref;
  }

  //-------------------------------------------------------------
---------------------
  // Constructors
  //-------------------------------------------------------------
---------------------
  /**
   * Construction d'un produit sans référence
   * @return une instance de Produit
   */
  public Produit() {
    _ref = null;
  }
  /**
   * Construction d'un produit avec une référence
   * @param ref la référence du produit
   * @return une instance de Produit
   */
  public Produit(Integer ref) {
    this.setRef(ref);
  }

  //-------------------------------------------------------------
---------------------
  // Méthodes
  //-------------------------------------------------------------
---------------------
```

```java
  /**
   * Obtenir une representation chaîne du produit
   * @return une representation chaîne du produit
   */
  public String toString() {
    StringBuffer sb = new StringBuffer("");
    sb.append(_ref);
    return sb.toString();
  }
}
```

```java
package production;

/**
 * Titre :         Formation Java Pratique
 * Description :   Formation Java Pratique - Exercices
 * Copyright :     Dr. S. GAMOURA 2001
 * Société :       DR. S. GAMOURA
 * @author JRE
 * @version 2.0
 */

import java.util.*;

/**
 * Classe ABSTRAITE représentant le stock de produits
génériques
 */
public abstract class Stock {
  //-------------------------------------------------------------
---------------------
  // Constructeurs
  //-------------------------------------------------------------
---------------------
  /**
   * Création d'un stock
   * @return une instance de Stock
   */
  public Stock() {
  }

  //-------------------------------------------------------------
---------------------
  // Methods
  //-------------------------------------------------------------
---------------------

  /**
  * Retourne la quantité en stock d'un produit
  * à partir de sa référence
  * @return le stock du produit en question
  */
  public abstract int verifierStockRef(Integer ref);

  /**
   * Distribuer
   * @param quantité la quantité demandée
   * @param ref la référence d demandée
   */
```

```
  public abstract void distribuerRef(int quantity, Integer
ref) throws StockException;

  /**
   * Produit le plus cher
   */
  public abstract Object plusCherProduit();

}
```

```
package production;

/**
 * Titre :         Formation Java Pratique
 * Description :   Formation Java Pratique - Exercices
 * Copyright :     Dr. S. GAMOURA 2001
 * Société :       DR. S. GAMOURA
 * @author ADO
 * @version 1.0
 */

import java.util.*;
import java.text.*;

/**
 * Exception de type StockException
 */
public class StockException extends Exception {
  //--------------------------------------------------------------
---------------------
  // Members
  //--------------------------------------------------------------
---------------------
  private Date _creationDate;

  //--------------------------------------------------------------
---------------------
  // Constructors
  //--------------------------------------------------------------
---------------------
  /**
   * Création d'une exception StockException
   * @param message le message d'erreur
   * @param creationDate la date de création
   * @return une instance de StockException
   */
  public StockException(String message, Date creationDate) {
    super(message);
    _creationDate = creationDate;
  }

  //--------------------------------------------------------------
---------------------
  // Acessors/Modifiers
  //--------------------------------------------------------------
---------------------
  /**
   * Retourne la date de création de l'erreur
   * @return une date
```

```
   */
  public String getDate() {
    return DateFormat.getInstance().format(_creationDate);
  }
}
```

```
package production.caveVins;

/**
 * Titre :         Formation Java Pratique
 * Description :   Formation Java Pratique - Exercices
 * Copyright :     Dr. S. GAMOURA 2001
 * Société :       DR. S. GAMOURA
 * @author JRE
 * @version 1.0
 */

import java.util.*;
import java.sql.*;

import production.*;
import utils.*;

/**
 * Classe représentant le stock de vins
 */
public class StockVin extends Stock {
  //------------------------------------------------------------
--------------------
  // Members
  //------------------------------------------------------------
--------------------
  /**
   * L'accès à la BD
   */
  private AccesBD _accesBD;

  //------------------------------------------------------------
--------------------
  // Constructeurs
  //------------------------------------------------------------
--------------------
  /**
   * Création d'un stock de vin
   * @return une instance de Stock
   */
  public StockVin() {
    try {
      _accesBD = new AccesBD();
    } catch (Exception e) {
      System.out.println("PB lors de la connexion à la BD : "
+ e.getMessage());
    }
  }

  //------------------------------------------------------------
--------------------
  // Methods
```

```
  //------------------------------------------------------------
---------------------
  /**
   * Récupération du nombre de bouteilles en stock à partir
d'une référence
   * @param ref la référence de la bouteille
   * @return le nombre de bouteilles en stock
   */
  public int verifierStockRef(Integer ref) {
    Vector results = null;
    try {
      // Exécution de la requête SQL
      results = _accesBD.selectionner("stock", "cave_a_vin",
"ref=" + ref.toString());
      //On enlève la ligne avec les noms de colonne
      results.remove(0);
    } catch (Exception e) {
      System.out.println("PB lors de la recherche du stock : "
+ e.getMessage());
    }
    //On retourne le résultat
    return ((Integer) ((Vector)
results.firstElement()).firstElement()).intValue();
  }

  /**
   * Distribuer
   * @param quantite la quantité demandée
   * @param ref la référence d demandée
   */
  public void distribuerRef(int quantity, Integer ref) throws
StockException {
    int qte = verifierStockRef(ref);
    if (qte >= quantity) {
      // On distribue les bouteilles demandés
      try {
        //Diminution de la quantité de stock
        qte = qte - quantity;
        // Exécution de la requête SQL :
        _accesBD.modifier("CAVE_A_VIN", "stock=" + qte, "ref="
+ ref);
      } catch (Exception e) {
        System.out.println("PB lors de la mise à jour du stock
: " + e.getMessage());
      }
    } else
      throw new StockException("Distribution impossbile.
Quantité en stock < Quantité demandée", new java.util.Date());
  }

  /**
   * Bouteille la plus chère
   */
  public Object plusCherProduit() {
    Vector chateaux = this.chateaux();
    Vin vin = null;
    for (int i = 0; i < chateaux.size(); i++) {
      String nomChateau = (String) chateaux.elementAt(i);
      Vector vins = this.vins(nomChateau);
      if (!vins.isEmpty()) {
        if (vin == null) {
```

```
          vin = (Vin) vins.firstElement();
        }
        for (int j = 1; j < vins.size(); j++) {
          Vin aVin = (Vin) vins.elementAt(j);
          if (aVin.estPlusCher(vin))
            vin = aVin;
        }
      }
    }
    return vin;
  }

  /**
   * Retourne la liste des noms de chateaux
   * @return la liste des noms de chateuax
   */
  public Vector chateaux() {
    Vector result = new Vector();
    try {
      //Exécution de la requête de sélection
      Vector tmp = _accesBD.selectionner("nom_du_chateau",
"CAVE_A_VIN");
      //Suppression de la 1ère ligne contenant le nom des
colonnes
      tmp.remove(0);
      //Mise "à plat" du résultat
      for (int i = 0; i < tmp.size(); i++) {
        String nom = ((Vector)
tmp.elementAt(i)).firstElement().toString();
        if (!result.contains(nom))
          result.addElement(nom);
      }
    } catch (SQLException e) {
      System.out.println("Erreur lors de la commande SQL");
    } finally {
      return result;
    }
  }

  /**
   * Retourne les vins pour un nom de chateau
   * @return les vins
   */
  public Vector vins(String nomChateau) {
    Vector result = new Vector();
    try {
      //Les champs à récupérer dans la base
      String champs =
"ref,nom_du_chateau,appellation,categorie,type,millesime,prix_
vente";
      //on double les apostrophes dans le nom du chateau pour
ne pas avoir de problèmes
      String nom = Tools.doublerApostrophes(nomChateau);
      //Exécution de la requete
      Vector tmpVector = _accesBD.selectionner(champs,
"CAVE_A_VIN", "nom_du_chateau='" + nom + "'");
      tmpVector.removeElementAt(0);
      //Création d'un objet Vin pour chaque résultat de la
requête
      for (int i = 0; i < tmpVector.size(); i++) {
        Vin unVin = new Vin((Vector) tmpVector.elementAt(i));
```

```
        result.addElement(unVin);
      }
    } catch (SQLException e) {
      System.out.println("Erreur lors de la commande SQL pour 
le nom de chateau : " + nomChateau);
    } finally {
      return result;
    }
  }

  /**
   * Rechercher une bouteille de vin dans
   * le stock à partir de sa référence
   * @param ref la référence de la bouteille de vin à trouver
   */
  public Vin getVin(Integer ref) {
    Vin vin = null;
    try {
      // Exécution de la requête SQL
      String champs = 
"ref,nom_du_chateau,appellation,categorie,type,millesime,prix_
vente";
      Vector results = _accesBD.selectionner(champs, 
"CAVE_A_VIN", "ref=" + ref);
      //On enlève la ligne avec les noms de colonne
      results.remove(0);
      //Création du vin avec les attributs récupérés
      vin = new Vin((Vector) results.firstElement());
    } catch (Exception e) {
      System.out.println("PB lors de la recherche du stock : " 
+ e.getMessage());
    }
    return vin;
  }

  /**
   * Rechercher des bouteilles de vin dans
   * le stock à partir d'une référence et/ou d'un domaine 
et/ou d'un type
   * @param ref la référence de la bouteille de vin à trouver
   */
  public Vector getVins(String ref, String nomChateau, String 
type) {
    Vector results = new Vector();
    String criteres = "";
    //On vérifie quels sont les critères renseignés
    if (nomChateau != null && !nomChateau.equals("")) {
      criteres += " nom_du_chateau='" + nomChateau + "'";
    }
    if (ref != null && !ref.equals("")) {
      if (!criteres.equals(""))
        criteres += " and ref=" + ref;
      else {
        criteres += " ref=" + ref;
      }
    }
    if (type != null && !type.equals("")) {
      if (!criteres.equals(""))
        criteres += " and type='" + type + "'";
      else
        criteres += " type='" + type + "'";
```

```
    }

    try {
      // Exécution de la requête SQL
      String champs =
"ref,nom_du_chateau,appellation,categorie,type,millesime,prix_
vente";
      Vector tmpVector = _accesBD.selectionner(champs,
"CAVE_A_VIN", criteres);
      //On enlève la ligne avec les noms de colonne
      tmpVector.remove(0);
      //Création d'un objet Vin pour chaque résultat de la
requête
      for (int i = 0; i < tmpVector.size(); i++) {
        //Création du vin avec les attributs récupérés
        Vin unVin = new Vin((Vector) tmpVector.elementAt(i));
        results.addElement(unVin);
      }
    } catch (Exception e) {
      System.out.println("PB lors de la recherche du stock : "
+ e.getMessage());
    }
    return results;
  }

  //-------------------------------------------------------------
---------------------
  // Quelques tests
  //-------------------------------------------------------------
---------------------
  public static void main(String[] args) {
    // Création d'un objet StockVin
    StockVin stk = new StockVin();

    //Liste des chateaux de la base
    Vector chateaux = stk.chateaux();
    System.out.println(chateaux);
    System.out.println("TOTAL : " + chateaux.size() + " noms
de chateaux");

    //Liste des vins du chateau Bellevue la Foret
    String nomChateau = "Bellevue la Foret";
    Vector vins = stk.vins(nomChateau);
    for (int i = 0; i < vins.size(); i++)
      System.out.println(vins.elementAt(i).toString());
    System.out.println("TOTAL : " + vins.size() + " bouteilles
de " + nomChateau);

    // Vérifier le stock de Balland
    System.out.println("Stock de bouteilles de Balland : " +
stk.verifierStockRef(new Integer(1272)));

    // Distribuer 10 bouteilles de Balland
    try {
      stk.distribuerRef(10, new Integer(1272));
    } catch (StockException ste) {
      System.out.println(ste.getMessage());
    }

    // Vérifier le stock de Balland
```

```
    System.out.println("Stock de bouteilles de Balland : " +
stk.verifierStockRef(new Integer(1272)));

    //Produit le plus cher
    System.out.println("Produit le plus cher : " +
stk.plusCherProduit());
  }
}
```

```
package production.caveVins;

/**
 * Titre :         Formation Java Pratique
 * Description :  Formation Java Pratique - Exercices
 * Copyright :    Dr. S. GAMOURA 2001
 * Société :      DR. S. GAMOURA
 * @author JRE
 * @version 2.0
 */

import production.*;
import java.util.*;

/**
 * Classe représentant un vin stocké dans la cave
 */
public class Vin extends Produit {
  //-------------------------------------------------------------
---------------------
  // Members
  //-------------------------------------------------------------
---------------------
  /** Nom du chateau */
  private String _nomDuChateau;
  /** Appelation du vin */
  private String _appellation;
  /** Catégorie du vin */
  private String _categorie;
  /** Type du vin (rosé, rouge, blanc) */
  private String _type;
  /** Millesime du vin */
  private Integer _millesime;
  /** Prix de vente du vin */
  private Double _prixVente;

  //-------------------------------------------------------------
---------------------
  // Acessors/Modifiers
  //-------------------------------------------------------------
---------------------
  /**
   * Retourne le nom du chateau du vin
   * @return _nomDuChateau
   */
  public String getNomDuChateau() {
    return _nomDuChateau;
  }
  /**
   * Met à jour le nom du chateau du vin
   * @param nomDuChateau
```

```
 */
public void setNomDuChateau(String nomDuChateau) {
  _nomDuChateau = nomDuChateau;
}

/**
 * Retourne l'appellation
 * @return _appellation
 */
public String getAppellation() {
  return _appellation;
}
/**
 * Met à jour l'appellation
 * @param appellation
 */
public void setAppellation(String appellation) {
  _appellation = appellation;
}

/**
 * Retourne la categorie
 * @return _categorie
 */
public String getCategorie() {
  return _categorie;
}
/**
 * Met à jour la categorie
 * @param categorie
 */
public void setCategorie(String categorie) {
  _categorie = categorie;
}

/**
 * Retourne le type
 * @return _type
 */
public String getType() {
  return _type;
}
/**
 * Met à jour le type
 * @param type
 */
public void setType(String type) {
  _type = type;
}

/**
 * Retourne le millesime
 * @return _millesime
 */
public Integer getMillesime() {
  return _millesime;
}
/**
 * Met à jour le millesime
 * @param millesime
 */
```

```
  public void setMillesime(Integer millesime) {
    _millesime = millesime;
  }

  /**
   * Retourne le prix de vente
   * @return _prixVente
   */
  public Double getPrixVente() {
    return _prixVente;
  }
  /**
   * Met à jour le prix de vente
   * @param prixVente
   */
  public void setPrixVente(Double prixVente) {
    _prixVente = prixVente;
  }

  //--------------------------------------------------------------
---------------------
  // Constructors
  //--------------------------------------------------------------
---------------------
  /**
   * Construction par défaut d'un vin
   * @param ref
   * @param nomDuChateau
   * @param appellation
   * @param categorie
   * @param type
   * @param millesime
   * @param prix_vente
   * @return une instance de Vin
   */
  public Vin(Integer ref, String nomDuChateau, String
appelation, String categorie, String type, Integer millesime,
Double prix_vente) {
    super(ref);
    this.setNomDuChateau(nomDuChateau);
    this.setAppellation(appelation);
    this.setCategorie(categorie);
    this.setType(type);
    this.setMillesime(millesime);
    this.setPrixVente(prix_vente);
  }

  /**
   * Construction d'un vin à partir d'un vecteur d'attributs
   * @param listeAttributs
   * @return une instance de Vin
   */
  public Vin(Vector listeAttributs) {
    this(
      (Integer) listeAttributs.elementAt(0),
      (String) listeAttributs.elementAt(1),
      (String) listeAttributs.elementAt(2),
      (String) listeAttributs.elementAt(3),
      (String) listeAttributs.elementAt(4),
      (Integer) listeAttributs.elementAt(5),
      (Double) listeAttributs.elementAt(6));
```

```
  }

  //------------------------------------------------------------
----------------------
  // Méthodes
  //------------------------------------------------------------
----------------------
  //------------------------------------------------------------
----------------------
  // Interface Comparable
  //------------------------------------------------------------
----------------------
  /**
   * Comparer deux bouteilles de vins
   * @param vin la bouteille de vin
   * @return true si la première bouteille de vin est plus
chère sinon false
   */
  public boolean estPlusCher(Object o) {
    return (this.getPrixVente().compareTo(((Vin)
o).getPrixVente()) > 0);
  }

  /**
   * Obtenir une representation chaîne du vin
   * @return une representation chaîne du vin
   */
  public String toString() {
    StringBuffer sb = new StringBuffer("");
    sb.append(super.toString()).append(" :
").append(_nomDuChateau).append("
").append(_prixVente).append(" Euros");
    return sb.toString();
  }
}
```

```
package utils;

/**
 * Titre :         Formation Java Pratique
 * Description :   Formation Java Pratique - Exercices
 * Copyright :     Dr. S. GAMOURA 2001
 * Société :       DR. S. GAMOURA
 * @author JRE
 * @version 1.0
 */

import java.sql.*;
import java.util.*;
import java.io.*;

/**
 * Classe permettant l'accès à la base de données par JDBC
 */
public class AccesBD {

  //------------------------------------------------------------
----------------------
  // Members
```

```
  //--------------------------------------------------------------
---------------------
  /** Connexion à la base de données */
  private Connection _connexion;
  /** Propriétés de connexion */
  private Properties _props;

  //Les valeurs par défaut :
  //************************
  // Chaîne identifiant le driver JDBC-ODBC
  private final static String JDBC_DRIVER =
"sun.jdbc.odbc.JdbcOdbcDriver";
  // URL de la base de données : par défaut celle des vins
  private final static String JDBC_URL = "jdbc:odbc:vins";
  // Paramètre de connexion : utilisateur
  private final static String JDBC_USER = "";
  // Paramètre de connexion : mot de passe
  private final static String JDBC_PASSWORD = "";

  //--------------------------------------------------------------
---------------------
  // Constructors
  //--------------------------------------------------------------
---------------------
  /**
   * Constructeur chargeant le pont JDBC-ODBC
   */
  public AccesBD() throws Exception {
    //Load des données du fichier de propriétés
    try {
      _props = new PropertiesUtil(this);
    } catch (IOException e) {
      System.out.println("Pb Impossible de charger le fichier
de propriété :" + e.getMessage());
    }

    //Récupération du driver
    String driver = _props.getProperty("connection.driver");
    if (driver == null)
      driver = JDBC_DRIVER;

    try {
      //Mise en place du driver
      Class.forName(driver);
    } catch (ClassNotFoundException ee) {
      ee.printStackTrace();
      new Exception("Problème d'accès aux données");
    }

    //Connection à la base
    this.connecter();
  }

  //--------------------------------------------------------------
---------------------
  // Méthodes
  //--------------------------------------------------------------
---------------------
  /**
   * Etablissement d'une connexion avec la base de données
```

```
   * @exception SQLException Si la connexion à la base de
données échoue
   */
  public void connecter() throws SQLException {
    //Récupération de la propriété URL
    String url = _props.getProperty("connection.url");
    if (url == null)
      url = JDBC_URL;

    //Récupération de la propriété LOGIN
    String login = _props.getProperty("connection.login");
    if (login == null)
      login = JDBC_USER;

    //Récupération de la propriété PASSWORD
    String password =
_props.getProperty("connection.password");
    if (password == null)
      password = JDBC_PASSWORD;

    // Connexion à la base de données
    _connexion = DriverManager.getConnection(url, login,
password);
    // Suppression de la validation automatique
    _connexion.setAutoCommit(false);
  }

  /**
   * Sélection d'un ensemble de données d'une table, après
connexion
   * @param champs Liste des champs à renvoyer séparés par des
virgules
   * @param table Nom de la table où effectuer la sélection
   * @return L'ensemble des données trouvées sous la forme
d'un vecteur de
   *          vecteurs, dont le premier contient le nom des
champs
   * @exception SQLException Si une erreur SQL se produit
   */
  public Vector selectionner(String champs, String table)
throws SQLException {
    return selectionner(champs, table, "");
  }

  /**
   * Sélection d'un ensemble de données d'une table
   * @param champs Liste des champs à renvoyer séparés par des
virgules
   * @param table Nom de la table où effectuer la sélection
   * @param criteres Critères de sélection (clause WHERE)
   * @return L'ensemble des données trouvées sous la forme
d'un vecteur de
   *          vecteurs, dont le premier contient le nom des
champs
   * @exception SQLException Si une erreur SQL se produit
   */
  public Vector selectionner(String champs, String table,
String criteres) throws SQLException {
    return selectionner(champs, table, criteres, "");
  }
```

```
  /**
   * Sélection d'un ensemble de données d'une table
   * @param champs Liste des champs à renvoyer séparés par des
virgules
   * @param table Nom de la table où effectuer la sélection
   * @param criteres Critères de sélection (clause WHERE)
   * @param tri Tri des données (clause ORDER BY)
   * @return L'ensemble des données trouvées sous la forme
d'un vecteur de
   *         vecteurs, dont le premier contient le nom des
champs
   * @exception SQLException Si une erreur SQL se produit
   */
  public Vector selectionner(String champs, String table,
String criteres, String tri) throws SQLException {
    // Vecteur contenant le résultat de la requête
    Vector results = new Vector();
    // Vecteur représentant les types des champs retournés
    Vector types;
    // Vecteur de travail représentant les valeurs de
l'enregistrement retourné
    Vector record;

    // Requête à exécuter
    Statement stmt;
    // Chaîne SQL représentant la requête à exécuter
    String query;
    // Jeu d'enregistrements retourné par la requête
    ResultSet rs;

    // Définition de la requête à partir des paramètres
    query = "SELECT DISTINCT " + champs + " FROM " + table;
    if (!criteres.equals(""))
      query += " WHERE " + criteres;
    if (!tri.equals(""))
      query += " ORDER BY " + tri;

    // Exécution de la requête
    stmt = _connexion.createStatement();
    rs = stmt.executeQuery(query);

    // Récupération des "méta-données" renvoyés
    // (nombre de colonnes et types des colonnes)
    ResultSetMetaData rsmd = rs.getMetaData();
    types = new Vector();
    record = new Vector();
    for (int i = 1; i <= rsmd.getColumnCount(); i++) {
      record.addElement(rsmd.getColumnName(i));
      types.addElement(new Integer(rsmd.getColumnType(i)));
    }
    results.addElement(record);

    // Ajout d'un vecteur de valeurs au vecteur résultat
    while (rs.next()) {
      record = new Vector();
      // Récuparation de chaque donnée selon son type
      for (int i = 1; i <= rsmd.getColumnCount(); i++) {
        switch (((Integer) types.get(i - 1)).intValue()) {
          case Types.VARCHAR :
            record.addElement(rs.getString(i));
            break;
```

```
            case Types.INTEGER :
              record.addElement(new Integer(rs.getInt(i)));
              break;
            case Types.SMALLINT :
              record.addElement(new Integer(rs.getShort(i)));
              break;
            case Types.DOUBLE :
              record.addElement(new Double(rs.getDouble(i)));
          }
        }
        results.addElement(record);
      }

      // Fermeture du jeu d'enregistrements
      rs.close();
      // Fermeture de la requête
      stmt.close();

      // Retour des résultats
      return results;
    }

    /**
     * Modification de champs dans une table
     * @param table Nom de la table où effectuer la modification
     * @param champs_valeurs Liste des champs avec les nouvelles
valeurs (du type : "stock=20")
     * @param criteres Critères de sélection (clause WHERE)
     * @return Le nombre de lignes modifiées
     * @exception SQLException Si une erreur SQL se produit
     */
    public int modifier(String table, String champs_valeurs,
String criteres) throws SQLException {
      //Nb de modifications
      int retour = 0;
      // Requête à exécuter
      Statement stmt;
      // Chaîne SQL représentant la requête à exécuter
      String query;

      // Définition de la requête à partir des paramètres
      query = "UPDATE " + table + " SET " + champs_valeurs;
      if (!criteres.equals(""))
        query += " WHERE " + criteres;

      // Exécution de la requête
      stmt = _connexion.createStatement();
      retour = stmt.executeUpdate(query);

      // Fermeture de la requête
      stmt.close();
      //Commit
      _connexion.commit();

      // Retour du résultat
      return retour;
    }

    /**
     * Insertion de champs dans une table
     * @param table Nom de la table où effectuer l'insertion
```

```
     * @param champs Liste des champs
     * @param valeurs Liste de valeurs insérées
     * @return Le nombre de lignes insérées
     * @exception SQLException Si une erreur SQL se produit
     */
    public int inserer(String table, String champs, String
valeurs) throws SQLException {
      //Nb d'insertions
      int retour = 0;
      // Requête à exécuter
      Statement stmt;
      // Chaîne SQL représentant la requête à exécuter
      String query;

      // Définition de la requête à partir des paramètres
      if (!champs.equals(""))
        champs = "(" + champs + ")";
      query = "INSERT INTO " + table + " " + champs + " VALUES
(" + valeurs + ")";

      // Exécution de la requête
      stmt = _connexion.createStatement();
      retour = stmt.executeUpdate(query);

      // Fermeture de la requête
      stmt.close();
      //Commit
      _connexion.commit();

      // Retour du résultat
      return retour;
    }

    /**
     * Suppression de lignes dans une table
     * @param table Nom de la table où effectuer la suppression
     * @param criteres Critères de sélection (clause WHERE)
     * @return Le nombre de lignes supprimées
     * @exception SQLException Si une erreur SQL se produit
     */
    public int supprimer(String table, String criteres) throws
SQLException {
      //Nb de modifications
      int retour = 0;
      // Requête à exécuter
      Statement stmt;
      // Chaîne SQL représentant la requête à exécuter
      String query;

      // Définition de la requête à partir des paramètres
      query = "DELETE * FROM " + table;
      if (!criteres.equals(""))
        query += " WHERE " + criteres;

      // Exécution de la requête
      stmt = _connexion.createStatement();
      retour = stmt.executeUpdate(query);

      // Fermeture de la requête
      stmt.close();
      //Commit
```

```
    _connexion.commit();

    // Retour du résultat
    return retour;
  }

  //-------------------------------------------------------------
---------------------
  // MAIN
  //-------------------------------------------------------------
---------------------
  /**
   * QUELQUES TESTS
   */
  public static void main(String[] args) {

    //SELECTION
    Vector res = null;
    try {
      AccesBD a = new AccesBD();
      res =
a.selectionner("ref,nom_du_chateau,prix_vente,stock",
"CAVE_A_VIN", "ref=1272");
    } catch (Exception e) {
      e.printStackTrace();
    }
    System.out.println(res);

    //UPDATE
    int nb = 0;
    try {
      AccesBD a = new AccesBD();
      nb = a.modifier("CAVE_A_VIN", "stock=stock-1",
"ref=1272");
    } catch (Exception e) {
      e.printStackTrace();
    }
    System.out.println(nb + " modification(s)");

    //SELECTION
    try {
      AccesBD a = new AccesBD();
      res =
a.selectionner("ref,nom_du_chateau,prix_vente,stock",
"CAVE_A_VIN", "ref=1272");
    } catch (Exception e) {
      e.printStackTrace();
    }
    System.out.println(res);
  }
}
```

accesBD.properties :

```
connection.driver=sun.jdbc.odbc.JdbcOdbcDriver
connection.url=jdbc:odbc:vins
connection.login=
connection.password=
```

```
package utils;

/**
 * Titre :        Formation Java Pratique
 * Description :  Formation Java Pratique - Exercices
 * Copyright :    Dr. S. GAMOURA 2001
 * Société :      DR. S. GAMOURA
 * @author JRE
 * @version 1.0
 */


import java.util.*;
import javax.naming.*;
import javax.naming.directory.*;
import java.io.*;


/**
 * Classe réalisant l'authentification d'un utilisateur à
partir d'un
 * login et d'un mot de passe, en utilisant l'interface JNDI
sur DSML
 */
public class Authentification {

  public final static int LOGIN_OK = 0;
  public final static int LOGIN_ERROR = 1;
  public final static int PASSWORD_ERROR = 2;

      //VALEURS PAR DEFAUT (si le fichier de propriétés n'est
pas trouvé)
      //*******************
  // Répertoire par défaut dans lequel se trouve la liste des
mots de passe
  private final static String DEFAULT_DIR = "c:/temp/";
  // Fichier DSML par défaut dans lequel se trouve la liste
des mots de passe
  private final static String DEFAULT_FILE =
"authentification.xml";

  // Contexte JNDI utilisé pour lire le fichier
d'authentification DSML
  private DirContext ctx;


  //-----------------------------------------------------------
---------------------
  // Constructeurs
  //-----------------------------------------------------------
---------------------
  /**
   * Constructeur vide
   * @exception NamingException Si l'initialisation du
contexte JNDI se passe mal
   */
  public Authentification() throws NamingException {
      Properties props = null;
```

```
      //Load des données du fichier de propriétés
    try {
                  props = new PropertiesUtil(this);
    } catch (IOException e) {
      System.out.println("Pb Impossible de charger le fichier
de propriété :" + e.getMessage());
    }

    //Initialisation des données en fonction des propriétés du
fichier
    String dir = props.getProperty("authentification.dir");
    if (dir == null)
                  dir = DEFAULT_DIR;
    String file = props.getProperty("authentification.file");
    if (file == null)
                  file = DEFAULT_FILE;

    this.initDSML(dir + file);

  }


  //---------------------------------------------------------------
---------------------
  // Méthodes
  //---------------------------------------------------------------
---------------------
  /**
   * Initialisation du contexte JNDI à partir du fichier
d'authentification DSML
   * @param Chemin d'accès complet vers le fichier
d'authentification DSML
   * @exception NamingException Si l'initialisation du
contexte JNDI se passe mal
   */
  private void initDSML(String authentificationPath) throws
NamingException {
      Hashtable env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.dsml.DsmlCtxFactory");
      env.put(Context.PROVIDER_URL, "file:/" +
authentificationPath);
      ctx = new InitialDirContext(env);
  }

  /**
   * Vérification de la concordance login-password
   * @param login Login de l'utilisateur
   * @param password Mot de passe de l'utilisateur
   * @return LOGIN_OK, LOGIN_ERROR ou PASSWORD_ERROR
   */
  public int authenticate(String login, String password) {
    String reponse = "";
    try {
      reponse = ctx.getAttributes("uid=" + login + ", ou=paca,
o=decan").
      get("passwd").get().toString();
    }
    catch (NameNotFoundException nnfex) {
      return LOGIN_ERROR;
    }
```

```
    catch (NamingException nex) {
      nex.printStackTrace();
      return LOGIN_ERROR;
    }
    // vérifie la validité du password
    if(password.equals(reponse))
      return LOGIN_OK;
    else
      return PASSWORD_ERROR;
  }
}
```

```
authentification.properties :

authentification.dir=c:/temp/
authentification.file=authentification.xml
```

```
package utils;

/**
 * Titre :         Formation Java Pratique
 * Description :   Formation Java Pratique - Exercices
 * Copyright :     Dr. S. GAMOURA 2001
 * Société :       DR. S. GAMOURA
 * @author JRE
 * @version 1.0
 */

import java.io.*;
import java.util.Properties;

/**
 * Classe permettant l'accès à un fichier de propriétés
 */
public class PropertiesUtil extends Properties {

  //-----------------------------------------------------------
---------------------
  // Members
  //-----------------------------------------------------------
---------------------
  /** Le nom du fichier */
      String _fileName=null;


  //-----------------------------------------------------------
---------------------
  // Constructeurs
  //-----------------------------------------------------------
---------------------
      /**
       * Constructeur à partir d'une classe aClass
       * Load les propriétés à partir du fichier
"aClass.properties"
       * @param aClass une class
       * @return une instance de PropertiesUtil loadée
       */
      public PropertiesUtil(Class aClass) throws IOException {
```

```
            super();
            //Création du nom du fichier de propriétés ==> 
dans le répertoire du user du Système
            _fileName = new String(
                  System.getProperty("user.home")
                  +System.getProperty("file.separator")
                  +"ressources"
                  +System.getProperty("file.separator")

      +(aClass.getName()).replace('.',System.getProperty("file
.separator").charAt(0))
                  +".properties"
            );

            //Création du flux d'entrée sur le fichier
            FileInputStream in = null;
            File fic = new File(_fileName);
            if (!fic.exists()) {
                  //Si le fichier ,n'existe pas
                  //1 - on crée l'arborescence de répertoire
                  fic.getParentFile().mkdirs();
                  //2 - on crée le fichier
                  fic.createNewFile();
            }
            in = new FileInputStream(_fileName);

            //load des propriétés du fichier
            this.load(in);
      }

      /**
       * Constructeur à partir d'un objet anObject
       * Load les propriétés à partir du fichier 
aClass.properties
       * @param aObject un objet
       * @return une instance de PropertiesUtil loadée
       */
      public PropertiesUtil(Object anObject) throws 
IOException,FileNotFoundException {
            this(anObject.getClass());
      }


  //-----------------------------------------------------------
---------------------
  // MAIN
  //-----------------------------------------------------------
---------------------
      /**
       * La méthode main pour les tests
       */
      public static void main(String[] args) {
            try {
                  PropertiesUtil prop = new 
PropertiesUtil(Class.forName("utils.AccesBD"));
            } catch (Exception e) {
                  System.out.println("Erreur lors du 
chargement des propriétés : "+e.toString());
            }
      }
}
```

```java
package utils;

/**
 * Titre :        Formation Java Pratique
 * Description :  Formation Java Pratique - Exercices
 * Copyright :    Dr. S. GAMOURA 2001
 * Société :      DR. S. GAMOURA
 * @author JRE
 * @version 1.0
 */

import java.util.*;

/**
 * Classe d'outils divers
 */
public class Tools {
  /**
   * Doublage des apostrophes dans une chaine
   * @param chaine la chaine à traiter
   * @return la nouvelle chaine
   */
  public static String doublerApostrophes(String chaine) {
    StringBuffer buf = new StringBuffer();
    char[] chars = chaine.toCharArray();
    for (int i = 0; i < chars.length; i++) {
      buf.append(chars[i]);
      if (chars[i] == '\'') {
        buf.append(chars[i]);
      }
    }
    return buf.toString();
  }
}
```

INDEX.HTML :

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<HTML>
      <HEAD>
            <META name="GENERATOR" content="IBM WebSphere
Studio">
            <TITLE>index.html</TITLE>
      </HEAD>
      <BODY background="images/fondvigne.gif">
<CENTER>
<TABLE border="1">
         <TR>
            <TD align="center" width="500" valign="top"
height="183"><IMG src="images/btf_vign.jpg" width="230"
height="100" border="0">
            <HR width="75%">
            <CENTER>
            <H1><A href="login.html">Bienvenue</A></H1>
            </CENTER>
            </TD>
        </TR>
```

```
</TABLE>
</CENTER>
<CENTER></CENTER>
</BODY>
</HTML>
```

NEWCOMMANDE.JSP :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<!-- SAisie d'une référence et d'un nom de client pour une
nouvelle commande -->

<%@ page session = "true" %>

<html>
<HEAD>
  <title>Nouvelle commande</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>

<BODY bgcolor="#DDDDDD" link="#000000" vlink="#000000"
alink="#000000" background="images/fondvigne.gif">

<center>
      <!-- commande-->
      <table border=1 width=500 height=300>
            <tr align=center valign=middle><td><IMG
src="images/btf_cata.jpg" width="230" height="100" border="0">
            <HR width="75%">
            <h1>Nouvelle commande</h1>
            <form method="post" action="liste.jsp">
            <table>
               <tr><td> Référence </td><td><input type="text"
name="newRef" size="25"></td></tr>
               <tr><td> Client </td><td><input type="text"
name="customer" size="25"></td></tr>
               <tr><td>  </td><td> </td></tr>
               <tr><td>   </td><td><input type="submit"
value="Submit"><input type="reset" value="Reset"> </td></tr>
                   <tr><td>   </td><td>   </td></tr>
                   <tr><td>   </td><td>   </td></tr>
        </table>
            </form>
      </td></tr>
      </table>
</center>

</body>
</html>
```

LISTE.JSP :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<!-- * Affichage de la liste de tous les noms de domaines de
la cave -->
```

```
<!-- * Choix d'un domaine par l'utilisateur -->
<!-- * Récapitulatif de la commande en cours -->
<!-- * Lancement de l'enregistrement de la commande -->

<%@ page import = "commande.*" %>
<%@ page import = "production.caveVins.StockVin" %>
<%@ page import = "java.util.*" %>

<%
      if (request.getParameter("newRef") != null) {
            // On entre pour la premiere fois ici : on crée
une nouvelle commande
            String ref = (String)
request.getParameter("newRef");
            String cust = (String)
request.getParameter("customer");
          Caddy caddy = (Caddy) session.getAttribute("caddy");
          caddy.setCommande(new Commande(ref,cust));
      }

      // Récupération de la liste des noms de domaines
      StockVin stk = new StockVin();
      Vector chateaux = stk.chateaux();
%>

<html>
<HEAD>
<title>Liste des domaines</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>

<BODY bgcolor="#DDDDDD" link="#000000" vlink="#000000"
alink="#000000" background="images/fondvigne.gif">

<!-- Affichage des noms de domaines de la cave -->
<CENTER>
<FORM method="get" action="choix.jsp">
<TABLE border="1">
    <TBODY>
        <TR>
            <TD valign="top" align="center" width="500"
height="300"><IMG src="images/btf_cata.jpg" width="230"
height="100" border="0">
            <HR width="75%">
            <H1>Domaines</H1>
            <SELECT name="choix">
                <%
                for (int i = 0; i < chateaux.size(); i++)
                {
%>
                <OPTION value="<%= chateaux.elementAt(i)
%>"><%= chateaux.elementAt(i) %> <%
                }
%></OPTION>
            </SELECT><BR>
            <BR>
            <INPUT type="image" src="images/raisigif.gif"
alt="Submit"></TD>
        </TR>
    </TBODY>
</TABLE>
```

```
</FORM>
<br>
<br>
<br>
<br>

<!-- Récapitulatif de la commande en cours --> <%
      Caddy caddy = (Caddy) session.getAttribute("caddy");
      Vector lignes = caddy.getCommande().getLignesCommande();
%>
<TABLE border=1 width=500>
    <TR><%="<H2>Commande en cours :
"+caddy.getCommande().getRef()+" ==>
"+caddy.getCommande().getCustomer()+"</H2>"%></TR>
    <TR>
        <TD><b>Produit</b></TD>
        <TD><b>Quantité</b></TD>
    </TR>
    <%
      for (int i = 0; i < lignes.size(); i++) {
            LigneCommande lc = (LigneCommande)
lignes.elementAt(i);
%>
    <TR>
        <td><%= lc.getName()%></td>
        <td><%= lc.getQuantity()%></td>
    </TR>
    <%
        }
%>
</TABLE>


<!-- Formulaire qui lance l'enregistrement de la commande -->
<form method="post" action="enregistrer.jsp"><input
type="submit" value="Enregistrer"></form>

</CENTER>
</body>
</html>
```

CHOIX.JSP :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<!-- Affichage de tous les vins pour le nom de domaine en
paramètre de la reqête -->
<!-- Choix d'une référence et d'une quantité par l'utilisateur
-->

<%@ page import = "production.caveVins.*" %>

<%@ page import = "java.util.*" %>
<%@ page import = "java.text.*" %>

<%
      //Affichage de tous les vins pour le nom de domaine en
paramètre de la reqête
      StockVin stk = new StockVin();
```

```
      String domaine = (String) request.getParameter("choix");
      Vector vins = stk.vins(domaine);
%>


<html>
<HEAD>
  <title>Domaine <%= domaine%></title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>

<BODY bgcolor="#DDDDDD" link="#000000" vlink="#000000"
alink="#000000" background="images/fondvigne.gif">
<P align="center"><IMG src="images/btf_cata.jpg" width="230"
height="100" border="0">
            </P>
<HR width="75%" align="center">
<BR>
<center>
    <table border=1>
      <tr align=center valign=middle>
            <td><b>Référence</b></td>
            <td><b>Appellation</b></td>
            <td><b>Catégorie</b></td>
            <td><b>Type</b></td>
            <td><b>Millesime</b></td>
            <td><b>Prix de vente</b></td>
      </tr>

      <%
            //Création d'un format d'affichage de nombres à
virgule
            DecimalFormat format = new DecimalFormat();
            format.setMaximumFractionDigits(2);
          format.setMinimumFractionDigits(2);

            for (int i = 0; i <vins.size();i++) {
                  //Pour chaque vin on affiche toutes ses
données
      %>
          <tr>
            <td><%=((Vin) vins.elementAt(i)).getRef()%></td>
            <td><%=((Vin)
vins.elementAt(i)).getAppellation()%></td>
            <td><%=((Vin)
vins.elementAt(i)).getCategorie()%></td>
            <td><%=((Vin) vins.elementAt(i)).getType()%></td>
            <td><%=((Vin)
vins.elementAt(i)).getMillesime()%></td>
            <td><%=format.format(((Vin)
vins.elementAt(i)).getPrixVente().doubleValue())%></td>
          </tr>
      <%
            }
      %>

    </table>

    <br>
<!-- choix de la quantité désirée -->
      <table border=1 width=500 height=300>
```

```
            <tr align=center valign=middle><td>
            <h1>Choix de la commande</h1>
            <form method="post" action="ajout.jsp">
            <table>
               <tr><td> Référence </td><td><input type="text"
name="ref" size="25"></td></tr>
               <tr><td> Quantity </td><td><input type="text"
name="quantity" size="25"></td></tr>
               <tr><td>  </td><td> </td></tr>
               <tr><td>   </td><td><input type="submit"
value="Submit"><input type="reset" value="Reset"> </td></tr>
                   <tr><td>   </td><td>   </td></tr>
                   <tr><td>   </td><td>   </td></tr>
        </table>
            </form>
      </td></tr>
      </table>

</center>

</body>
</html>
```

AJOUT.JSP :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<!-- Ajout d'une nouvelle ligne de commande -->

<%@ page import = "commande.Caddy" %>
<%@ page import = "production.*" %>
<%@ page import = "production.caveVins.*" %>

<%@ page import = "java.util.*" %>

<%
      //Ajout du nombre de bouteilles voulues dans la commande
en cours
      boolean ajout=true;
      Integer ref = new Integer((String)
request.getParameter("ref"));
      Integer quantity = new Integer((String)
request.getParameter("quantity"));
      Caddy caddy = (Caddy) session.getAttribute("caddy");
      StockVin stk = new StockVin();
      try {
            //Diminution du stock
            stk.distribuerRef(quantity.intValue(), ref);
            //ajout d'une ligne de commande à la commande en
cours

      caddy.add(stk.getVin(ref).toString(),quantity.intValue()
);
      } catch (StockException e) {
            ajout=false;
      } finally {
%>

<html>
```

```
<HEAD>
  <title>Ajout...</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>

<BODY bgcolor="#DDDDDD" link="#000000" vlink="#000000"
alink="#000000" background="images/fondvigne.gif">
<center>
<%
      if (ajout==true) {
%>
      <h1>Ajout effectué</h1>
<%
      } else {
%>
      <h1>Ajout impossible : il ne reste que
<%=stk.verifierStockRef(ref)%> bouteilles en stock !</h1>
<%
      }
%>
      <br><br><br>
      <a href="liste.jsp" alt="Continuer"><img
src="images/raisigif.gif"></a>
</center>
</body>
</html>
<%
      }
%>
```

ENREGISTRER.JSP :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<!-- Enregistrement de la commande et export de toutes les
commandes au format CSV -->

<%@ page import = "commande.*" %>
<%@ page import = "java.util.*" %>

<%
      Caddy caddy = (Caddy) session.getAttribute("caddy");
      GererCommandes gestion = new GererCommandes();
      //Enregistrement de la commande dans la base
      gestion.enregistreCommande(caddy.getCommande());
      //Export de toutes les commandes de la base
      gestion.exportCSV();
%>

<html>
<HEAD>
  <title>Enregitrement effectué</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>

<BODY bgcolor="#DDDDDD" link="#000000" vlink="#000000"
alink="#000000" background="images/fondvigne.gif">
      <center>
      <h1>Enregistrement de la commande effectué</h1>
```

```
      <br><br><br>
      <a href="newCommande.jsp" alt="Nouvelle commande"><img
src="images/raisigif.gif"></a>
    </center>
</body>
</html>
```

TESTACCESBD.JAVA :

```
import junit.framework.TestCase;
import java.util.*;

public class TestAccesBD extends TestCase {

      public TestAccesBD(String name) {
         super(name);
      }

      public void testSelection() {
        //  SELECTION
        Vector res = null;
        try {
            AccesBD a = new AccesBD();
            res =
a.selectionner("ref,nom_du_chateau,prix_vente,stock",
"CAVE_A_VIN", "ref=1272");
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println(res);
        assertTrue(res.size() > 0);
      }

      public void testMaj() {
        //UPDATE
        int nb = 0;
        try {
            AccesBD a = new AccesBD();
            nb = a.modifier("CAVE_A_VIN", "stock=stock-1",
"ref=1272");
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println(nb + " modification(s)");
        assertEquals(nb, 1);
      }

      public static void main(String[] args) {
                  junit.textui.TestRunner.run(new
TestAccesBD(""));
            }

}
```