```
package util;

import au.com.bytecode.opencsv.CSVReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import metier.service.Service;

/**
 * La classe LectureDonneesCsv permet (comme son nom l'indique) la lecture de
 * données CSV dans des fichiers. Elle doit être complétée et personnalisée pour
 * interagir avec VOTRE couche service pour la création effective des entités.
 * Elle comprend initialement la lecture d'un fichier Clients et d'un fichier
 * Pays. Une méthode {@link main()} permet de tester cette classe avant de
 * l'intégrer dans le reste de votre code.
 *
 * @author Équipe DASI - 2013/2014
 */
public class LectureDonneesCsv {

    public static final String cheminFichier = "C:\\Users\\Ordi\\Desktop\\INSA"
            + "\\3IF\\TP_DASI\\DASI-master\\Projets\\data\\";
    public static final String cheminFichier1 = "/Users/KEV/NetBeansProjects/"
            + "GitHub/DASI/Projets/data/";
    public static final String fichierClients = cheminFichier
            + "IFRoutard-Clients.csv";
    public static final String fichierPays = cheminFichier
            + "IFRoutard-Pays.csv";
    public static final String fichierDeparts = cheminFichier
            + "IFRoutard-Departs.csv";
    public static final String fichierCircuits = cheminFichier
            + "IFRoutard-Voyages-Circuits.csv";
    public static final String fichierSejours = cheminFichier
            + "IFRoutard-Voyages-Sejours.csv";
    public static final String fichierConseillers = cheminFichier
            + "IFRoutard-Conseillers.csv";

    public static int NBLIGNES = -1;

    public static int NBCLIENTS = -1;
```

```
    /**
     * Format de date pour la lecture des dates dans les fichiers CSV fournis.
     */
    protected static DateFormat CSV_DATE_FORMAT
            = new SimpleDateFormat("yyyy-MM-dd");

    /**
     * Format de date pour l'affichage à l'écran.
     */
    protected static DateFormat USER_DATE_FORMAT
            = new SimpleDateFormat("dd/MM/yyyy");

    /**
     * Le lecteur de fichier CSV. Il doit être initialisé avant l'appel aux
     * méthodes de la classe.
     */
    protected CSVReader lecteurFichier;

    /**
     * Unique constructeur de la classe. Le fichier CSV donné en paramètre doit
     * avoir le point-virgule ';' comme séparateur et être encodé en UTF-8. Le
     * fichier est immédiatement ouvert (en lecture) par ce constructeur.
     *
     * @param cheminVersFichier Chemin vers le fichier CSV.
     * @throws FileNotFoundException Si le chemin vers le fichier n'est pas
     * valide ou le fichier non-lisible.
     */
    public LectureDonneesCsv(String cheminVersFichier) throws
            FileNotFoundException, UnsupportedEncodingException {

        this.lecteurFichier = new CSVReader(
                new InputStreamReader(
                        new FileInputStream(cheminVersFichier), "UTF-8"), ';');
    }

    /**
     * Ferme le fichier CSV. Les autres méthodes ne doivent plus être appelées
     * après cela.
     *
     * @throws IOException
     */
    public void fermer() throws IOException {


        this.lecteurFichier.close();
    }
```

```
    /**
     * Méthode statique pour lire une date à partir d'une chaîne de caractère.
     * Adapté au format de date des fichiers CSV fournis, par exemple:
     * 2014-02-01.
     *
     * @param date Chaîne de caractère représentant la date.
     * @return La date interpétée ou la date actuelle en cas mauvais format en
     * entrée.
     */
    protected static Date parseDate(String date) {
        try {
            return CSV_DATE_FORMAT.parse(date);
        } catch (ParseException ex) {
            return new Date();
        }
    }

    /**
     * Méthode statique pour formater une date pour l'affichage. Par exemple:
     * 01/02/2014.
     *
     * @param date Date à formater.
     * @return Chaîne de caractère représentant la date.
     */
    protected static String formatDate(Date date) {
        return USER_DATE_FORMAT.format(date);
    }

    /**
     * Méthode statique pour afficher l'en-tête d'un fichier CSV lu par le
     * lecteur. L'affichage se fait sur la "sortie d'erreur" (en rouge dans la
     * console sous Netbeans). Le nom des colonnes est précédé de leur index
     * dans le tableau (commençant à 0).
     *
     * @param colonnes le tableau des noms de colonnes.
     */
    protected static void afficherEnTeteCsv(String[] colonnes) {

        for (int i = 0; i < colonnes.length; i++) {
            if (i > 0) {

                System.err.print("; ");
            }
            System.err.print("#" + Integer.toString(i) + " => " + colonnes[i]);
        }
```

```
        System.err.println();
    }

    /**
     * Lit le fichier CSV, affiche son en-tête, puis appelle la création de
     * Client pour chaque ligne.
     *
     * @param limite Nombre maximum de lignes à lire ou -1 pour ne pas limiter
     * @throws IOException
     */
    public void lireClients(int limite) throws IOException {

        String[] nextLine;

        // En-tete du fichier CSV
        nextLine = this.lecteurFichier.readNext();
        afficherEnTeteCsv(nextLine);

        // Lecture des lignes
        while ((nextLine = this.lecteurFichier.readNext()) != null) {

            creerClient(nextLine);

            // Limite (ou -1 si pas de limite)
            if (!(limite < 0) && (--limite < 1)) {
                break;
            }
        }

    }

    /**
     * Créée un Client à partir de sa description. La date de naissance est
     * notamment interpétée comme un objet Date.
     *
     * @param descriptionClient Ligne du fichier CSV de Clients.
     */
    public void creerClient(String[] descriptionClient) {

        String civilite = descriptionClient[0];

        String nom = descriptionClient[1];
        String prenom = descriptionClient[2];
        String dateNaissance = descriptionClient[3];
        String adresse = descriptionClient[4];
        String telephone = descriptionClient[5];
        String email = descriptionClient[6];
```

```
        String codevoyagepourdevis = descriptionClient[7];

        Service.creerClient(civilite, nom, prenom, dateNaissance, adresse,
                telephone, email);
        Service.creerDevis(codevoyagepourdevis, email);
    }

    /**
     * Lit le fichier CSV, affiche son en-tête, puis appelle la création de Pays
     * pour chaque ligne.
     *
     * @param limite Nombre maximum de lignes à lire ou -1 pour ne pas limiter
     * @throws IOException
     */
    public void lirePays(int limite) throws IOException {

        String[] nextLine;

        // En-tete du fichier CSV
        nextLine = this.lecteurFichier.readNext();
        afficherEnTeteCsv(nextLine);

        // Lecture des lignes
        while ((nextLine = this.lecteurFichier.readNext()) != null) {

            creerPays(nextLine);

            // Limite (ou -1 si pas de limite)
            if (!(limite < 0) && (--limite < 1)) {
                break;
            }
        }

    }

    /**
     * Créée un Pays à partir de sa description. La superficie et la population
     * sont notamment interpétées comme des nombres.

     *
     * @param descriptionClient Ligne du fichier CSV de Pays.
     */
    public void creerPays(String[] descriptionPays) {

        String nom = descriptionPays[0];
        String code = descriptionPays[1];
```

```
        String region = descriptionPays[2];
        String capitale = descriptionPays[3];
        String langues = descriptionPays[4];
        Float superficie = Float.parseFloat(descriptionPays[5]);
        Float population = Float.parseFloat(descriptionPays[6]);
        String regime = descriptionPays[7];

        Service.creerPays(nom, code, region, capitale, langues, superficie,
                population, regime);

    }

    /**
     * Lit le fichier CSV, affiche son en-tête, puis appelle la création de
     * Conseiller pour chaque ligne.
     *
     * @param limite Nombre maximum de lignes à lire ou -1 pour ne pas limiter
     * @throws IOException
     */
    public void lireConseiller(int limite) throws IOException {

        String[] nextLine;

        // En-tete du fichier CSV
        nextLine = this.lecteurFichier.readNext();
        afficherEnTeteCsv(nextLine);

        // Lecture des lignes
        while ((nextLine = this.lecteurFichier.readNext()) != null) {

            creerConseiller(nextLine);

            // Limite (ou -1 si pas de limite)
            if (!(limite < 0) && (--limite < 1)) {
                break;
            }
        }


    }

    /**
     * Créée un Conseiller à partir de sa description. La date de naissance est
     * notamment interpétée comme un objet Date.
     *
     * @param descriptionClient Ligne du fichier CSV de Clients.
     */
```

```
    public void creerConseiller(String[] descriptionConseiller) {

        String civilite = descriptionConseiller[0];
        String nom = descriptionConseiller[1];
        String prenom = descriptionConseiller[2];
        String dateNaissance = descriptionConseiller[3];
        String adresse = descriptionConseiller[4];
        String telephone = descriptionConseiller[5];
        String email = descriptionConseiller[6];

        String[] PaysConseilles = new String[descriptionConseiller.length - 7];
        for (int tailletab = 0; tailletab < descriptionConseiller.length - 7;
                tailletab++) {
            PaysConseilles[tailletab] = descriptionConseiller[tailletab + 7];
        }

        Service.creerConseiller(civilite, nom, prenom, dateNaissance, adresse,
                telephone, email, PaysConseilles);

    }

    public void lireDeparts(int limite) throws IOException {

        String[] nextLine;

        // En-tete du fichier CSV
        nextLine = this.lecteurFichier.readNext();
        afficherEnTeteCsv(nextLine);

        // Lecture des lignes
        while ((nextLine = this.lecteurFichier.readNext()) != null) {

            creerDeparts(nextLine);

            // Limite (ou -1 si pas de limite)

            if (!(limite < 0) && (--limite < 1)) {
                break;
            }
        }

    }

    public void creerDeparts(String[] descriptionDeparts) {

        String codeVoyage = descriptionDeparts[0];
```

```
        String date = descriptionDeparts[1];
        String ville = descriptionDeparts[2];
        int tarif = Integer.parseInt(descriptionDeparts[3]);
        String transport = descriptionDeparts[4];

        Service.creerInfoPrincipale(ville, date, tarif, transport, codeVoyage);

    }

    public void lireSejours(int limite) throws IOException {

        String[] nextLine;

        // En-tete du fichier CSV
        nextLine = this.lecteurFichier.readNext();
        afficherEnTeteCsv(nextLine);

        // Lecture des lignes
        while ((nextLine = this.lecteurFichier.readNext()) != null) {

            creerSejours(nextLine);

            // Limite (ou -1 si pas de limite)
            if (!(limite < 0) && (--limite < 1)) {
                break;
            }
        }

    }

    public void creerSejours(String[] descriptionSejours) {

        String codePays = descriptionSejours[0];
        String codeVoyage = descriptionSejours[1];

        String intitule = descriptionSejours[2];
        int duree = Integer.parseInt(descriptionSejours[3]);
        String description = descriptionSejours[4];
        String residence = descriptionSejours[5];

        Service.creerSejour(residence, codePays, codeVoyage, intitule, duree,
                description);

    }

    public void lireCircuits(int limite) throws IOException {
```

```
        String[] nextLine;

        // En-tete du fichier CSV
        nextLine = this.lecteurFichier.readNext();
        afficherEnTeteCsv(nextLine);

        // Lecture des lignes
        while ((nextLine = this.lecteurFichier.readNext()) != null) {

            creerCircuits(nextLine);

            // Limite (ou -1 si pas de limite)
            if (!(limite < 0) && (--limite < 1)) {
                break;
            }
        }

    }

    public void creerCircuits(String[] descriptionCircuits) {

        String codePays = descriptionCircuits[0];
        String codeVoyage = descriptionCircuits[1];
        String intitule = descriptionCircuits[2];
        int duree = Integer.parseInt(descriptionCircuits[3]);
        String description = descriptionCircuits[4];
        String transport = descriptionCircuits[5];
        int kilometres = Integer.parseInt(descriptionCircuits[6]);

        Service.creerCircuit(transport, kilometres, codePays, codeVoyage,
                intitule,
                duree, description);


    }

     public static void initClient(){
         initClient(-1);
     }
    public static void initClient(int nbClients) {

        setNBCLIENTS(nbClients);
        try {

            LectureDonneesCsv lectureDonneesCsv_Clients
                    = new LectureDonneesCsv(fichierClients);
```

```
            lectureDonneesCsv_Clients.lireClients(NBCLIENTS);

            lectureDonneesCsv_Clients.fermer();

        } catch (IOException ex) {
            ex.printStackTrace(System.err);
        }

    }

    public static void initPays()
    {
        initPays(-1);
    }
    public static void initPays(int nbLignes) {

        setNBLIGNES(nbLignes);
        try {

            LectureDonneesCsv lectureDonneesCsv_Pays
                    = new LectureDonneesCsv(fichierPays);

            lectureDonneesCsv_Pays.lirePays(NBLIGNES);

            lectureDonneesCsv_Pays.fermer();

        } catch (IOException ex) {
            ex.printStackTrace(System.err);
        }


    }

     public static void initDeparts() {
         initDeparts(-1);
     }
    public static void initDeparts(int nbLignes) {

        setNBLIGNES(nbLignes);
        try {

            LectureDonneesCsv lectureDonneesCsv_Departs
                    = new LectureDonneesCsv(fichierDeparts);

            lectureDonneesCsv_Departs.lireDeparts(NBLIGNES);
```

```
            lectureDonneesCsv_Departs.fermer();

        } catch (IOException ex) {
            ex.printStackTrace(System.err);
        }

    }

    public static void initCircuits() {
        initCircuits(-1);
    }
    public static void initCircuits(int nbLigne) {

        setNBLIGNES( nbLigne) ;
        try {

            LectureDonneesCsv lectureDonneesCsv_Circuits
                    = new LectureDonneesCsv(fichierCircuits);

            lectureDonneesCsv_Circuits.lireCircuits(NBLIGNES);

            lectureDonneesCsv_Circuits.fermer();

        } catch (IOException ex) {
            ex.printStackTrace(System.err);
        }

    }
    public static void initSejours( ) {

        initSejours(-1);
    }

    public static void initSejours( int nbLigne) {

        try {
    setNBLIGNES(nbLigne);
            LectureDonneesCsv lectureDonneesCsv_Sejours
                    = new LectureDonneesCsv(fichierSejours);

            lectureDonneesCsv_Sejours.lireSejours(NBLIGNES);

            lectureDonneesCsv_Sejours.fermer();

        } catch (IOException ex) {
            ex.printStackTrace(System.err);
```

```
        }

    }

    public static void initConseillers() {

        initConseillers(-1);

    }

public static void initConseillers( int nbLigne)
{
    setNBLIGNES(nbLigne);
            try {

            LectureDonneesCsv lectureDonneesCsv_Conseillers
                    = new LectureDonneesCsv(fichierConseillers);

            lectureDonneesCsv_Conseillers.lireConseiller(NBLIGNES);

            lectureDonneesCsv_Conseillers.fermer();

        } catch (IOException ex) {
            ex.printStackTrace(System.err);
        }

}


    public static void initAll() {
        initPays();
        initConseillers();
        initCircuits();
        initSejours();
        initDeparts();
        initClient();

    }

    public static void setNBLIGNES(int NBLIGNES) {
        LectureDonneesCsv.NBLIGNES = NBLIGNES;
    }

    public static void setNBCLIENTS(int NBCLIENTS) {
        LectureDonneesCsv.NBCLIENTS = NBCLIENTS;
    }
```

```
}
```