INFORMATIQUE

Semestre 2

UNIVERSITÉ DE NANTES

U.F.R. des Sciences et des Techniques

S.E.V.E. Bureau des Examens

Année universitaire 2014-2015
Semestre 2
Session 1

Nom de l'U.E. : Informatique
Code de l'U.E. : **L3 SEN-X6SE010**
Date de L'examen : Mai 2015
Durée : 2h00
Documents autorisés : OUI
Calculatrice autorisée : OUI

Examen d'informatique

Thème du sujet
Page web sur les règnes de rois de France.

Description générale de la page à écrire :
La fenêtre est divisée en 2 cadres utilisant la même feuille de style «**PresentationRois.css**» (voir annexe).
Les 3 fichiers html à écrire sont
«**DépartRois.html**», «**Index.html**» et «**Rois.html**». Le 1er fichier découpe la fenêtre en 2 cadres, le 2ème est affiché dans le cadre de gauche et le 3ème dans le cadre de droite (voir captures d'écran).

Fichier «Rois.html»
Le cadre de droite ne contient aucune fonction javascript. On y trouve le titre du document, l'image à l'ouverture se trouvant dans le fichier «**rois_de_france.jpg**» et un formulaire contenant une zone texte affichée sous l'image destinée à recevoir les dates de règne de chaque roi. Cette zone texte utilise le style défini par «Regne» dans la feuille de style. L'image sera changée par les événements utilisateur survenant dans le cadre de gauche.

Fichier « Index.html»
Le cadre de gauche propose à l'utilisateur de choisir, grâce à 3 menus déroulants, un roi de France. Chaque menu correspond à une dynastie.
Menu «**Valois**» : François 1er, Charles IX et Henri III
Menu «**Bourbons**» : Louis XIII, Louis XIV, Louis XV, Louis XVI et Louis XVIII
Menu «**Orléans**» : Louis Philippe
Le bouton «**Annuler**» remet les **deux** cadres dans l'état initial.
Le choix dans un des 3 menus affiche dans le cadre de droite l'image correspondant au roi sélectionné ainsi que ses dates de règnes dans la zone texte en dessous de l'image. Les noms des fichiers d'images sont construits en juxtaposant le nom de la dynastie, le numéro d'ordre dans le menu et .jpg.
Exemple «Bourbons2.jpg» contient l'image de Louis XIV

Questions :

Partie html

1. Écrire le code html du fichier «**DépartRois.html**»
2. Écrire le code html du fichier «**Rois.html**»
3. Écrire le code html du fichier « **Index.html**»

Partie javascript

On utilisera pour les dates de règnes les tableaux de chaînes de caractères suivants :

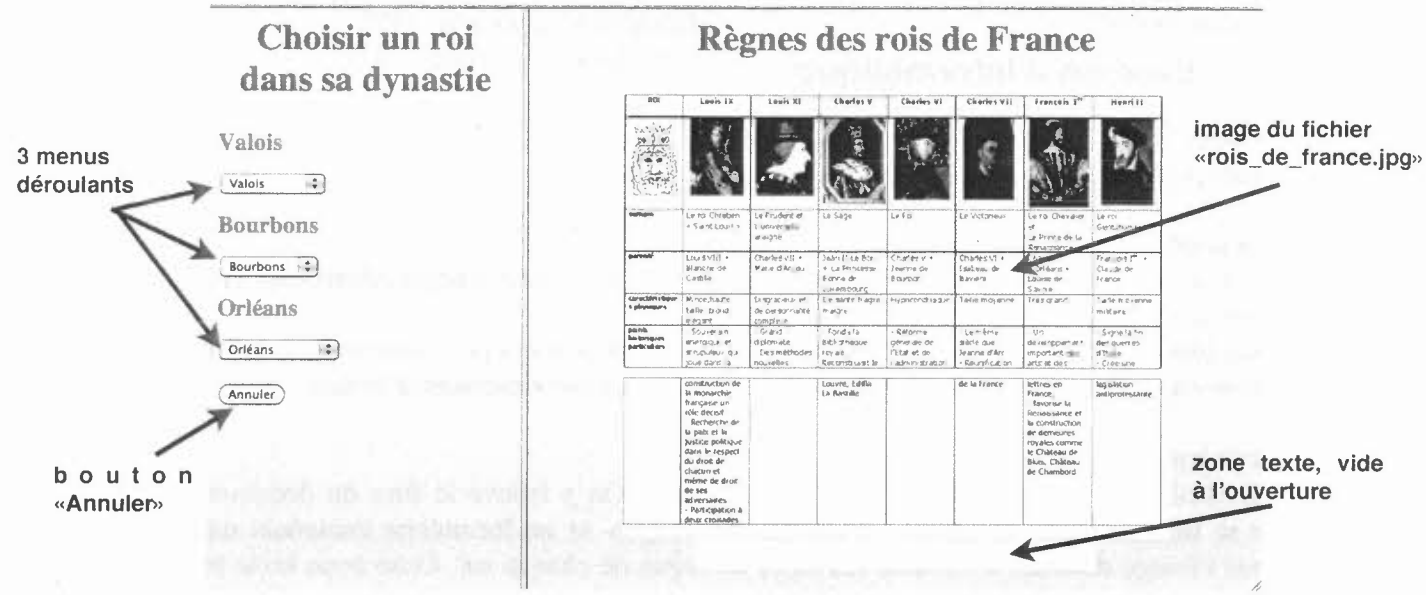
```
RegneValois=["","1515 à 1547","1560 à 1574","1574 à 1589"];
RegneBourbons=["","1610 à 1643 ","1643 à 1715","1715 à 1774", "1774 à 1792","1814 à 1824"];
RegneOrleans=["","1830 à 1848"];
```

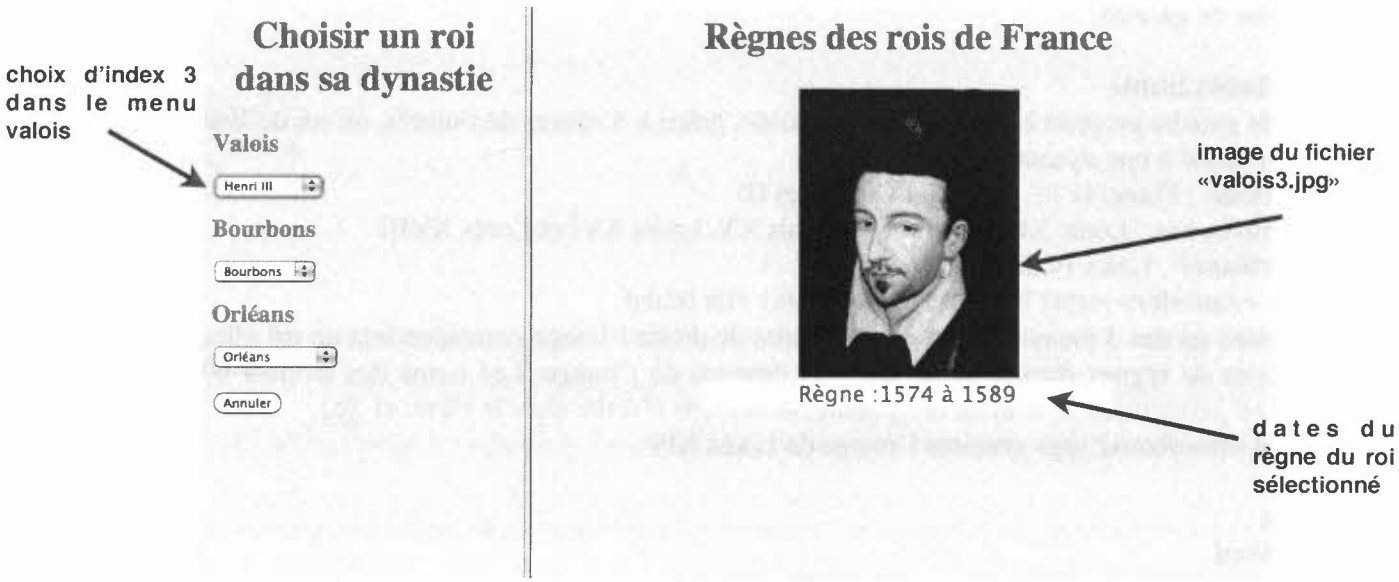
Les fonctions javascript à écrire doivent respecter les consignes suivantes :

- Lorsque les 3 menus ne présentent aucun choix de rois, l'image à droite est «rois_de_france.jpg»
- Les menus déroulants ne peuvent pas montrer simultanément des choix de rois de dynasties différentes.

4. Programmer le bouton «**Annuler**» (événement et fonction javascript)
5. Écrire le code javascript de la fonction **AfficheRoi(n)** qui affiche l'image et les dates de règne dans le cadre de droite. n est une variable contenant le numéro du menu dans lequel s'est fait le choix. Écrire les appels de cette fonction.

Captures d'écran - annexe





Feuille de style (fichier «PresentationRois.css»)

```
body {
        background-color: #FFE0BC;
        font-family: times;
        font-size: 12pt;
        }

h1 { text-align: left;
        color: #FF7A00;
        font-size: 16pt;
        }
h2 { text-align: center;
        color: #EE00FF;
        font-size: 24pt;
        }
```

```
.Regne {
        font-size: 14pt;
        color: #DD6600;
        background-color:#FFF7CA;
        border-style: solid;
        border-top-width: 1px;
        border-left-width: 1px;
        border-bottom-width: 1px;
        border-right-width:1px;
        border-color: #FFF7CA;
        text-align: center;
}
```

Session 2
Rattrapages

Licence Informatique
X5I0010

Université de Nantes
2014-2015

Examen de seconde session

Durée : 90 minutes.
Imprimés et notes de cours / TD / TP autorisés. Tout le reste est interdit.
Il n'est pas obligatoire de faire les exercices dans l'ordre.

Exercice 1 (Polymorphisme - 4 points)

Considérons le code Java suivant :

```
import java.io.*;

class A {
  /* A */
  public String method(double x, double y) {
    return "A";
  }

  /* B */
  public String method(int x, int y) {
    return "B";
  }
}

class B extends A {
  /* C */
  public String method(int x, double y) {
    return "C";
  }

  /* D */
  public String method(int x, int y) {
    return "D";
  }

  /* E */
  public String method(char x, char y) {
    return "E";
  }
}

class C extends B {
  /* F */
  public String method(char x, char y) {
    return "F";
  }
}

class Main {
  public static void main(String [] args) {
    A a = new C();
    System.out.println( a.method( '9', '9' ) );
  }
}
```

Sachant qu'un valeur de type `char` est automatiquement convertible vers les types `int` et `double`, et qu'une valeur de type `int` est automatiquement convertible vers le type `double`, écrivez le résultat affiché (A, B, C, D, E ou F) en justifiant votre réponse. Les détails de votre réponse doivent décrire ce qui se passe à la compilation (**2 points**) et lors de l'exécution (**2 points**).

Exercice 2 (Le lexique linguistique - 16 points)

Nous introduisons un système de gestion d'un lexique linguistique. Un lexique linguistique est, dans notre exemple, un ensemble de verbes et de noms. L'utilisateur ajoute les noms sous la forme de leur singulier. La forme plurielle est ajoutée automatiquement. L'utilisateur ajoute les verbes sous la forme de leur infinitif. Les formes conjuguées sont ajoutées automatiquement. La forme plurielle d'un nom et les formes conjuguées d'un verbe sont appelés ses formes fléchies.

Notre lexique linguistique est représenté par la classe `LexiqueLinguistique`. Celle-ci détient les listes des noms et des verbes.

Ces listes, ainsi que les données des formes fléchies, peuvent être libérées de la mémoire pour être enregistrées dans le fichier associé au lexique. Elles peuvent ensuite être rechargées en mémoire pour ainsi être relues.

Il existe également deux modes de fonctionnement :

- le mode de préparation des données dans lequel l'utilisateur peut ajouter des mots. En mode de préparation des données, quand l'utilisateur demande les formes fléchies d'un mot, elles sont calculées à ce moment-là et ne sont pas stockées en mémoire. (gain en espace, coût en temps) ;
- le mode d'exploitation des données où toutes les formes fléchies ont été calculées et enregistrées. En mode d'exploitation, quand l'utilisateur demande les formes fléchies d'un mot, elles sont directement lues en mémoire et retournées sans calcul (gain en temps, coût en espace). Dans ce mode, il n'est pas possible d'ajouter de nouveau mot.

La classe `LexiqueLinguistique` contient les attributs suivants, les méthode étant commentées dans le code :

- L'attribut `noms` est la liste des noms, l'attribut `verbes` la liste des verbes.
- L'attribut `formesFlechiesNom` contient, pour chaque nom, la liste des ses formes fléchies.
- L'attribut `formesFlechiesVerbe` contient, pour chaque verbe, la liste des ses formes fléchies.

Le code source de la classe `LexiqueLinguistique` est le suivant :

```
import java.io.*;
import java.util.*;

class LexiqueLinguistique
{
  protected List<String> noms = new ArrayList<String>();
  protected List<String> verbes = new ArrayList<String>();
  protected Map<String, List<String>> formesFlechiesNom
    = new HashMap<String, List<String>>();
  protected Map<String, List<String>> formesFlechiesVerbe
    = new HashMap<String, List<String>>();
  protected ByteArrayOutputStream fluxSortie = new ByteArrayOutputStream();

  // Par simplification, on fait abstraction des try/catch d'exceptions
  // pourtant preferables dans les methodes suivantes.

  /*
   * Enregistre les donnees des quatre attributs de la classe et les
   * libere de la memoire.
   */
```

```
protected void enregistreDonnees()
{
  fluxSortie.reset();
  ObjectOutputStream fluxObjet = new ObjectOutputStream(fluxSortie);
  fluxObjet.writeObject(noms);
  fluxObjet.writeObject(verbes);
  fluxObjet.writeObject(formesFlechiesNom);
  fluxObjet.writeObject(formesFlechiesVerbe);
}

/*
 * Relit en memoire les donnees qui ont ete enregistrees par enregistreDonnee.
 */
protected void litDonnees()
{
  ByteArrayInputStream fluxEntree
    = new ByteArrayInputStream(fluxSortie.toByteArray());
  ObjectInputStream fluxObjet = new ObjectInputStream(fluxEntree);
  noms = (List<String>) fluxObjet.readObject();
  verbes = (List<String>) fluxObjet.readObject();
  formesFlechiesNom = (HashMap<String, List<String>>) fluxObjet.readObject();
  formesFlechiesVerbe = (HashMap<String, List<String>>) fluxObjet.readObject();
}

/*
 * Ajoute un nom dans la liste des noms. Si jamais les donnees ont
 * ete liberees de la memoire, elles sont relues.  Si les attributs
 * formesFlechiesNom et formesFlechiesVerbe sont non vides, l'ajout
 * n'est pas possible (mode d'exploitation).
 *
 * Retourne vrai si et seulement si l'ajout a ete possible.
 *
 * La methode compareToIgnoreCase retourne 0 si et seulement si les
 * deux strings testees sont identiques, modulo la casse.
 */
public boolean ajouteNom(String leNom)
{
  if( noms == null )
    litDonnees();
  if( !formesFlechiesNom.isEmpty() || !formesFlechiesVerbe.isEmpty() )
    return false;
  for( String nom : noms )
    if( nom.compareToIgnoreCase( leNom ) == 0 )
      return false;
  noms.add( leNom );
  return true;
}

/*
 * Meme principe que ajouteNom, mais pour les verbes.
 */
public boolean ajouteVerbe(String leVerbe)
{
  if( verbes == null )
    litDonnees();
  if( !formesFlechiesNom.isEmpty() || !formesFlechiesVerbe.isEmpty() )
    return false;
  for( String verbe : verbes )
    if( verbe.compareToIgnoreCase( leVerbe ) == 0 )
      return false;
  verbes.add( leVerbe );
  return true;
}
```

```
/*
 * Libere toutes les donnees de la memoire en les enregistrant.
 */
public void libereMemoire()
{
  if( noms == null && verbes == null )
    return;
  enregistreDonnees();
}

/*
 * Calcule toutes les formes flechies. Il convient de noter que les
 * formes flechies sont ici calculees en anglais et de facon
 * tres simplifiees.  Le mode d'exploitation est active.
 */
public void modeExploitation()
{
  if( noms == null  && verbes == null )
    litDonnees();

  for( String nom : noms )
    formesFlechiesNom.put( nom, Arrays.asList( nom + "s" ) );

  for( String verbe : verbes )
    formesFlechiesVerbe.put( verbe, Arrays.asList( verbe,
                                                   verbe + "ed",
                                                   verbe + "ing" ) );
}

/*
 * Efface toutes les formes flechies calculees. Le mode de
 * preparation des donnees est active.
 */
public void modePreparation()
{
  if( noms == null  && verbes == null )
    litDonnees();
  formesFlechiesNom.clear();
  formesFlechiesVerbe.clear();
}

/*
 * Renvoie les formes flechies d'un nom. En mode de preparation des
 * donnees, elles sont calculees.  En mode d'exploitation, les formes
 * flechies sont obtenues a partir des donnees de formesFlechiesNom.
 */
public List<String> formesFlechiesNom(String leNom)
{
  if( noms == null )
    litDonnees();
  if( formesFlechiesNom.isEmpty() && formesFlechiesVerbe.isEmpty() )
    {
      for( String nom : noms )
        if( nom.compareToIgnoreCase( leNom ) == 0 )
          return Arrays.asList( nom + "s" );
      return null;
    }
  return formesFlechiesNom.get( leNom );
}

/*
 * Meme principe que formesFlechiesNom, mais pour les verbes.
```

```
   */
  public List<String> formesFlechiesVerbe(String leVerbe)
  {
    if( verbes == null )
      litDonnees();
    if( formesFlechiesNom.isEmpty() && formesFlechiesVerbe.isEmpty() )
      {
        for( String verbe : verbes )
          if( verbe.compareToIgnoreCase( leVerbe ) == 0 )
            return Arrays.asList( verbe, verbe + "ed", verbe + "ing" );
        return null;
      }
    return formesFlechiesVerbe.get( leVerbe );
  }
}
```

Le programme suivant permet de tester la classe `LexiqueLinguistique` :

```
import java.util.List;

class Test
{
  public static void main( String[] args )
  {
    List<String> resultatFormesFlechies;
    LexiqueLinguistique lexique = new LexiqueLinguistique();
    lexique.ajouteVerbe( "walk" );
    lexique.libereMemoire();

    resultatFormesFlechies = lexique.formesFlechiesVerbe( "walk" );
    if( resultatFormesFlechies != null )
      for( String forme : resultatFormesFlechies )
        System.out.println( forme );
    else
      System.out.println( "walk␣est␣un␣mot␣inconnu␣!" );

    lexique.ajouteVerbe( "pick" );
    lexique.modeExploitation();
    lexique.ajouteNom( "floor" );
    lexique.libereMemoire();

    resultatFormesFlechies = lexique.formesFlechiesVerbe( "pick" );
    if( resultatFormesFlechies != null )
      for( String forme : resultatFormesFlechies )
        System.out.println( forme );
    else
      System.out.println( "pick␣est␣un␣mot␣inconnu␣!" );

    resultatFormesFlechies = lexique.formesFlechiesNom( "floor" );
    if( resultatFormesFlechies != null )
      for( String forme : resultatFormesFlechies )
        System.out.println( forme );
    else
      System.out.println( "floor␣est␣un␣mot␣inconnu␣!" );
  }
}
```

L'exécution du programme fournit le résultat suivant :

```
walk
walked
walking
```

```
pick
picked
picking
floor est un mot inconnu !
```

1. Le code de la classe `LexiqueLinguistique` compile (si on ajoute les try/catch d'exceptions) et fonctionne correctement. Mais quelque chose ne va pas au niveau de la conception. Expliquer quel est le problème en vous justifiant. (**2 points**)
2. Proposer un pattern qui peut s'appliquer à la classe `LexiqueLinguistique` dans le but de la simplifier. Justifiez votre choix. (**4 points**)
3. Écrire en Java votre pattern (**8 points**), et modifier en conséquence la classe `LexiqueLinguistique` (**2 points**).