INFORMATIQUE

Semestre 2

UNIVERSITÉ DE NANTES

U.F.R. des Sciences et des Techniques

S.E.V.E. Bureau des Examens

Année universitaire 2013-2014
Semestre 2
Session 1

Nom de l'U.E. : Informatique
Code de l'U.E. : **L3 SEN-X6SE010**
Date de L'examen : 15 mai 2014
Durée : 2h00
Documents autorisés : OUI
Calculatrice autorisée : OUI

Examen d'informatique

Thème du sujet

Description de la page à réaliser

La page présente 2 cadres :

le cadre de gauche montre le fichier «Niveaux.html»
le cadre de droite montre le fichier «Resultats.html»

Ces deux fichiers partage la feuille de style «presentation.css». (voir annexe)

Le cadre de gauche

Il permet à l'utilisateur, à l'aide de 3 boutons, de se positionner dans le fichier du cadre de droite au bon endroit pour saisir les résultats (Poules, 1/2 finales et finale).

Le cadre de droite

Il permet de saisir les résultats des différents matchs et de calculer automatiquement les équipes qualifiées ou l'équipe vainqueur pour la finale. Les saisies des résultats de matchs se font dans des zones texte affichées à l'aide du style «**score**» de la feuille de style. Les zones de texte pour afficher les noms des pays participants utilisent le style «**pays**» de la feuille. Le fichier html contient 3 étiquettes «**poules**», «**demifinales**» et «**finale**» permettant de faire le positionnement lorsqu'on clique sur les boutons du cadre de gauche.

Questions

1. Écrire le code HTML du fichier découpant la page en 2 cadres.

2. Écrire le code HTML du fichier «**Niveaux.html**». **Aide** : les liens vers le cadre de droite peuvent se faire sans fonction javascript en mettant les boutons dans une ancre.

3. Écrire le code HTML du fichier «**Resultats.html**». On n'écrira pas la totalité des tables de poules mais seulement un modèle (par exemple la poule A avec une ligne type)

Partie javascript : <u>On ne fera aucun test de validité des données fournies par l'utilisateur.</u>

4. Écrire la fonction javascript **CalculVainqueur**() qui calcule et affiche le vainqueur de la finale quand on clique sur le bouton «**Valider**». Insérer l'appel selon le bon événement.

5. Écrire la fonction javascript **CalculFinale**() qui calcule et affiche les équipes qualifiées pour la finale quand on clique sur le bouton «**Valider**». Insérer l'appel selon le bon événement.

6. Donner une idée d'algorithme pour calculer les équipes de poules qualifiées pour les 1/2 finales. Chaque match gagné rapporte 3 points au vainqueur, 0 point au perdant. Un match nul rapporte 1 point à chaque équipe. Le premier d'une poule rencontre le second de l'autre poule en 1/2 finale.

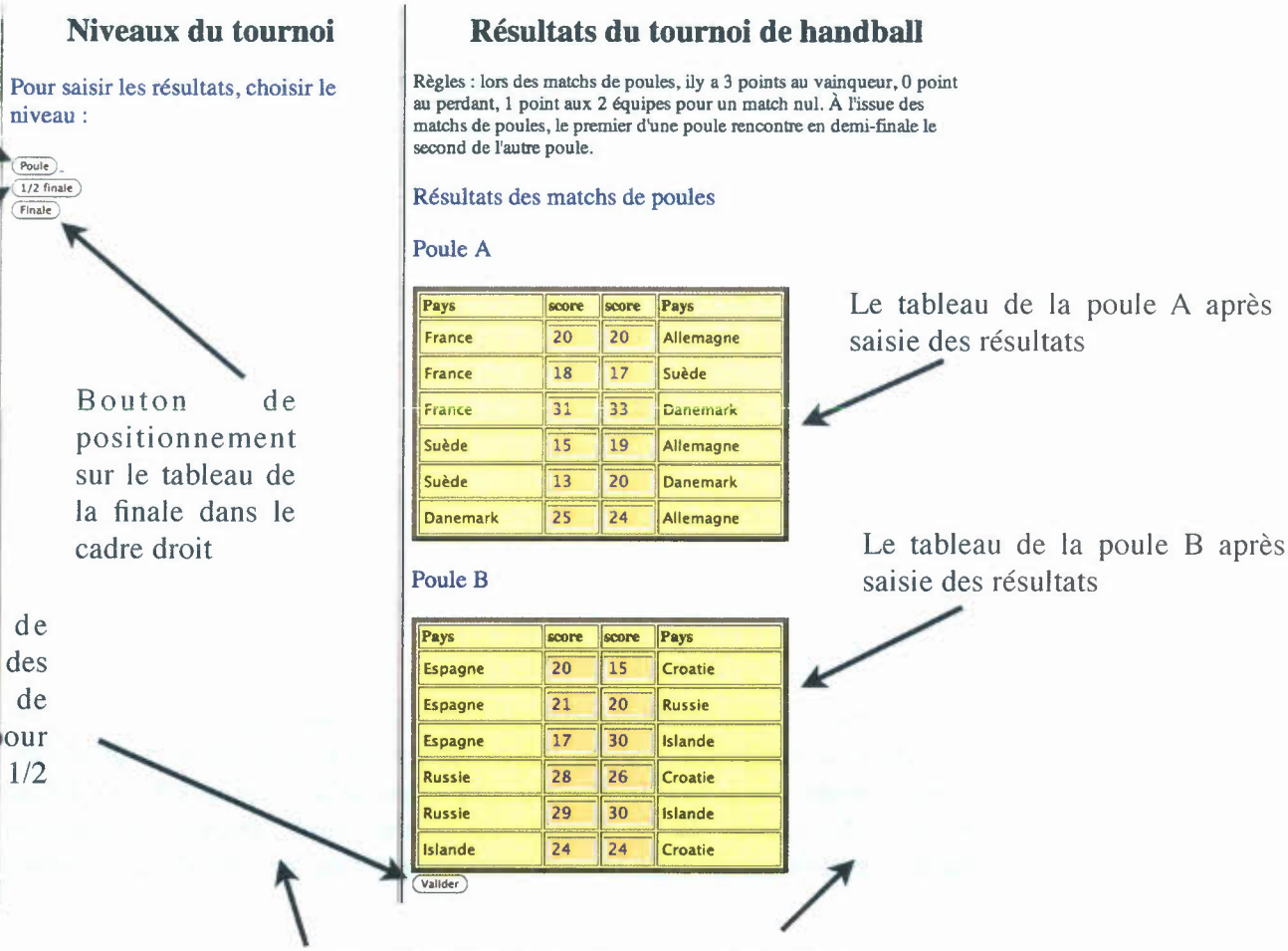
Captures d'écran - annexe

Bouton de positionnement sur les tableaux des poules dans le cadre droit

Bouton de positionnement sur le tableau des 1/2 finales dans le cadre droit

Bouton de positionnement sur le tableau de la finale dans le cadre droit

Bouton de validation des résultats de poules pour affichage des 1/2 finales



Le tableau de la poule A après saisie des résultats

Le tableau de la poule B après saisie des résultats

La page en 2 cadres à l'ouverture :
«Niveaux.html» à gauche et «Resultats.html» à droite

Fichier «Resultats.html»
avant validation des résultats de poules (à gauche) et après validation (à droite)

Fichier «Resultats.html»
avant validation des résultats des 1/2 finales (à gauche) et après validation (à droite)

Feuille de style (fichier «presentation.css»)

```
table {
        border-width: 5px;
        background-color: #FFFFAB;
        text-align: left;
        }
h1 {font-size: 24pt;
        text-align: center;}
body {
        font-size: 14pt;
        background-color: #D8FFFF;
        }
p {text-align: left;
        font-size: 18pt;
        color: blue;
                }
.score{
                font-size: 12pt;
                width:3em;
                color: #0000FF;
                background-color:#FFE19B;
                        }
.pays{
                font-size: 11pt;
                width:8em;
                color: #000000;
                background-color:#FFFFAB;
                border-style: solid;
                border-top-width: 0px;
                border-left-width: 0px;
                border-bottom-width: 0px;
                border-right-width:0px;
}
}
```

Session 2
Rattrapages

Licence Informatique
X5I0010

Université de Nantes
2013-2014

Examen

Durée : 1h30.
Imprimés et notes de cours / TD / TP autorisés. Tout le reste est interdit.
Libre à vous de prendre des raccourcis dans vos diagrammes UML et votre code Java.
C'est à vous de juger ce qui est "raccourcissable" ou pas.

Exercice 1 (La bibliothèque graphique)

Nous disposons d'une bibliothèque graphique capable de gérer des points. Elle se décline en trois types :

- 2D qui permet de gérer des points en deux dimensions en noir et blanc,
- 3D qui permet de gérer des points en trois dimensions en noir et blanc,
- 3DCouleur qui permet de gérer des points en trois dimensions en couleur.

Les points sont décrit par l'interface `Point` et les classes et sous-classes `Point2D`, `Point3D` et `Point3DCouleur` incluses dans le paquetage `dessin.bibliothequeGraphique`.

```
package dession.bibliothequeGraphique;

public interface Point {
  int getX();
  int getY();
  void setX( int x );
  void setY( int y );
}

public class Point2D implements Point {
  protected int x;
  protected int y;

  public Point2D( int x, int y ) {
    super();
    this.x = x;
    this.y = y;
  }

  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }

  public void setX( int x ) {
    this.x = x;
  }

  public void setY( int y ) {
    this.y = y;
  }
}
```

```
public class Point3D extends Point2D {
  protected int z;

  public Point3D( int x, int y, int z ) {
    super( x, y );
    this.z = z;
  }

  public int getZ() {
    return z;
  }

  public void setZ( int z ) {
    this.z = z;
  }
}

public class Point3DCouleur extends Point3D {
  protected int couleur;

  public Point3D( int x, int y, int z, int c ) {
    super( x, y, z );
    couleur = c;
  }

  public int getCouleur() {
    return couleur;
  }

  public void setCouleur( int c ) {
    couleur = c;
  }
}
```

La classe `Point3D` introduit l'attribut supplémentaire `z` et la class `Point3DCouleur` introduit l'attribut supplémentaire `couleur`.

Nous introduisons maintenant, dans le paquetage `dessin.formesGraphiques`, la classe abstraite `FormeGraphiquePlane` qui décrit une forme plane monochrome composée d'un ensemble quelconque de points. Cette classe est cliente de la bibliothèque graphique présentée ci-dessus.

Le code source de la classe `FormeGraphiquePlane` est présenté à la suite. Son constructeur prend trois paramètres :

- `typeBibliothequeGraphique` : le type de la bibliothèque graphique (2D, 3D ou 3DCouleur) qui sert à déterminer la classe des points qui composent la forme,
- `zPoint` : la valeur par défaut de la coordonnée `z` (la coordonnée `z` est la même pour tous les points des formes planes).
- `couleurPoint` : la valeur de la couleur par défaut (la couleur est la même pour tous les points des formes monochromes).

Sa méthode `ajoutePoint` ajoute un nouveau point à la forme. Le point créé dépend du type de la bibliothèque graphique qui a été passé comme paramètre au constructeur.

Sa méthode `dessine` est abstraite. Elle est implémentée dans les sous-classes concrètes de `FormeGraphiquePlane`.

```java
package dessin.formesGraphiques;

import java.util.*;
import dessin.bibliothequeGraphique.*;

public abstract class FormeGraphiquePlane {
  protected List<Point> points;
  protected String typeBibliothequeGraphique;
  protected int zPoint;
  protected int couleurPoint;

  public FormeGraphiquePlane() {
    points = new ArrayList<Point>();
    typeBibliothequeGraphique = "2D";
    zPoint = 0;
    couleurPoint = 0;
  }

  public FormeGraphiquePlane( String type, int z, int couleur ) {
    points = new ArrayList<Point>();
    typeBibliothequeGraphique = type;
    zPoint = z;
    couleurPoint = couleur;
  }

  public void ajoutePoint( int x, int y ) {
    Point nouveauPoint;

    if( typeBibliothequeGraphique.equals( "2D" ) ) {
      nouveauPoint = new Point2D( x, y );
    }
    else if( typeBibliothequeGraphique.equals( "3D" ) ) {
      nouveauPoint = new Point3D( x, y, zPoint );
    }
    else if( typeBibliothequeGraphique.equals( "3DCouleur" ) ) {
      nouveauPoint = new Point3DCouleur( x, y, zPoint, couleurPoint );
    }
    else {
      nouveauPoint = new Point2D( x, y );
    }

    points.add( nouveauPoint );
  }

  abstract public void dessine();
}
```

À l'intérieur du même paquetage, les deux sous-classes concrètes `Ligne` et `Triangle` implémentent la méthode **dessine** par une simulation comme le montre leur code source :

```java
package dessin.formesGraphiques;

public class Ligne extends FormeGraphiquePlane {
  public Ligne( String type, int z, int couleur,
                int x0, int y0, int x1, int y1 ) {
    super( type, z, couleur );
    ajoutePoint( x0, y0 );
    ajoutePoint( x1, y1 );
  }

  public void dessine() {
    System.out.println( "ligne␣de␣"
```

```
                        + points.get(0).getX() + ","
                        + points.get(0).getY() + " a "
                        + points.get(1).getX() + ","
                        + points.get(1).getY() );
  }
}

public class Triangle extends FormeGraphiquePlane {
  public Triangle( String type, int z, int couleur,
                   int x0, int y0, int x1, int y1, int x2, int y2 ) {
    super( type, z, couleur );
    ajoutePoint( x0, y0 );
    ajoutePoint( x1, y1 );
    ajoutePoint( x2, y2 );
  }

  public void dessine() {
    System.out.println( "triangle dont les sommets sont :");
    for( int i = 0; i < 3; i++ )
      System.out.println( "Sommet (" + (i+1) + ") = "
                          + points.get(i).getX() + ","
                          + points.get(i).getY() );
  }
}
```

Enfin, nous avons écrit le petit programme de test suivant qui créé un triangle constitué de points en 3D :

```
package dessin;

import dessin.formesGraphiques.Triangle;

public class Test {
  static void main( String[] args ) {
    Triangle triangle = new Triangle( "3D", 10, 0, 10, 10, 20, 20, 30, 30 );
    triangle.dessine();
  }
}
```

Son exécution fournit le résultat suivant :

```
triangle dont les sommets sont :
Sommet (1) = 10,10
Sommet (2) = 20,20
Sommet (3) = 30,30
```

Énoncé

Plusieurs défauts sont présents dans l'implémentation actuelle. Le but de l'exercice est de les retrouver, les commenter, et proposer un ou plusieurs patterns pour corriger chacun de ces défauts, de modéliser ce ou ces patterns en UML dans le contexte présenté ci-dessus et de les mettre en œuvre en modifiant en conséquence l'implémentation actuelle.