Nom Prénom :

Consignes relatives au déroulement de l'épreuve

Date :	17 juin 2016
Contrôle de :	**4IRC - Module « Programmation Orienté Objet » -**
Durée :	2h
Professeur responsable :	Françoise Perrin
Documents :	Supports de cours et TP autorisés Livres, calculatrice, téléphone, ordinateur et autres appareils de stockage de données numériques interdits.
Divers :	Les oreilles des candidats doivent être dégagées.

Rappels importants sur la discipline lors des examens

La présence à tous les examens est strictement obligatoire ; tout élève présent à une épreuve doit rendre une copie, même blanche, portant son nom, son prénom et la nature de l'épreuve.
Une absence non justifiée peut entrainer l'invalidation du module
Toute suspicion sur la régularité et le caractère équitable d'une épreuve est signalée à la direction des études qui pourra décider l'annulation de l'épreuve; tous les élèves concernés par l'épreuve sont alors convoqués à une épreuve de remplacement à une date fixée par le responsable d'année.
Toute fraude ou tentative de fraude est portée à la connaissance de la direction des études qui pourra réunir le Conseil de Discipline. Les sanctions prises peuvent aller jusqu'à l'exclusion définitive du (des) élève(s) mis en cause.

Recommandations

L'objectif de cette épreuve est de vérifier votre capacité à comprendre un programme existant et à le maintenir, et en particulier, vérifier que vous êtes capable :

De justifier ou critiquer certains choix de conception objet.
De justifier ou critiquer le choix de certaines structures de données.
De dire ce que font certaines instructions en expliquant leur intérêt pour le programme.
D'écrire des portions de code inexistantes.

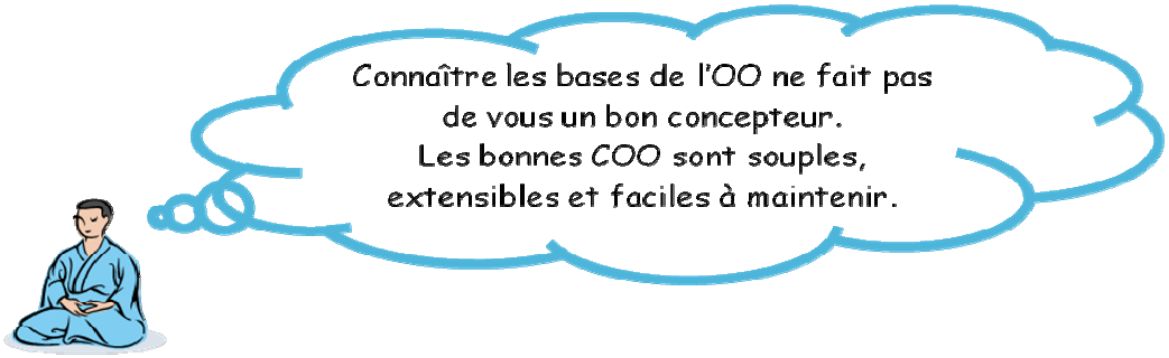
Lisez bien tout l'énoncé - y compris les annexes et trucs et astuces -

Les questions se rapportent au projet de jeu d'échec réalisé en séance

Les questions sont écrites en gras.

Pour les instructions Java, la syntaxe n'a pas d'importance pourvu qu'elle soit compréhensible.

Pour les questions qui attendent des réponses en français, ne récitez (ne recopiez ☹) pas le cours de manière théorique mais soyez **très près** de l'exemple. Soyez **synthétique** mais **précis**.



Rappels de conception du projet

La conception du projet a permis d'identifier un certain nombre de classes « métier » et en particulier :

- Des pièces : roi, reine, fous, pions, cavaliers, tours. Il existe 16 pièces blanches et 16 pièces noires.
- Des jeux : ensemble des pièces d'un joueur. Il existe 1 jeu blanc et 1 jeu noir.
- 1 échiquier qui contient les 2 jeux.

Chaque classe a ses propres responsabilités. Ainsi la classe `Jeu` est responsable de créer ses `Pieces` et de les manipuler. `L'Echiquier` quant à lui crée les 2 jeux mais ne peut pas manipuler directement les pièces. Pour autant, c'est lui qui est capable de dire si un déplacement est légal, d'ordonner ce déplacement, de gérer l'alternance des joueurs, de savoir si le roi est en échec et mat, etc. Pour ce faire, il passe donc par les objets `Jeu` pour communiquer avec les `Pieces`.

Les classes sont donc parfaitement bien encapsulées et les seules interactions possibles avec une IHM se font à travers `L'Echiquier` et en aucun cas une IHM ne pourra directement déplacer une `Pieces` sans passer par les méthodes de `L'Echiquier` (en fait à travers une classe `ChessGame`).

1. Questions de programmation sur le projet (8 points)

a) Soit la classe Coord suivante :

```
public class Coord implements Serializable {

   public int x, y;

   public Coord(int x, int y) {
         this.x = x;
         this.y = y;
   }
   public String toString() {
         return "[x=" + x + ", y=" + y + "]";
   }
   public static boolean coordonnees_valides(int a, int b){
         return ( (a<=7) && (a>=0) && (b<=7) && (b>=0) );
   }
   public int hashCode() {
         // …
   }
   public boolean equals(Object obj) {
         // …
   }
}
```

Pourquoi la méthode `coordonnees_valides()` est-elle déclarée avec le qualificateur « static » (Ne répondez pas « parce qu'elle est statique » ☺) **?**

b) Soit les interface et classe `Pieces` et `AbstractPiece` suivantes :

```
public interface Pieces {
   public int getX();
   public int getY();
   public Couleur getCouleur();
   public String getName();
   public boolean isMoveOk(int xFinal, int yFinal) ;
   public boolean move(int xFinal, int yFinal);
   public boolean capture();
};
public abstract class AbstractPiece implements Pieces {

   private int x, y;
   private Couleur couleur;

   public AbstractPiece(Couleur couleur, Coord coord){
          this.x = coord.x;
          this.y = coord.y;
          this.couleur=couleur;
   }
   public int getX(){
          return this.x;
   }
   public int getY(){
          return this.y;
   }
   public Couleur getCouleur(){
          return this.couleur;
   }
   public String getName(){
          return this.getClass().getSimpleName();
   }
   public boolean move(int x, int y){
          boolean ret = false;
          if(Coord.coordonnees_valides(x,y)){
                this.x=x; this.y=y;
                ret = true;
          }
          return ret;
   }
   public boolean capture(){
          this.x=-1; this.y=-1;
          return true;
   }
   public String toString(){
          String S = (this.getClass().getSimpleName()).substring(0, 2)
                       + " " + this.x + " " + this.y;
          return S;
   }
   public abstract boolean isMoveOk(int xFinal, int yFinal) ;
}
```

Pourquoi la méthode `isMoveOk()` est-elle déclarée avec le qualificateur « abstract » (Ne répondez pas « parce qu'elle est abstraite » ☺) **?**

c) Soit la classe `Jeu` suivante :

```
public class Jeu  {
   protected List<Pieces> pieces;
   protected Couleur couleur;

   public Jeu(Couleur couleur){
          this.pieces = ChessPiecesFactory.newPieces(couleur);
          this.couleur = couleur;
   }

   // Autres methodes publiques …

   private Pieces findPiece(int x, int y){
          Pieces pieceToFind = null;
          for (Pieces piece : pieces){
                if (piece.getX()==x && piece.getY()==y){
                      pieceToFind = piece;
                }
          }
          return pieceToFind;
   }
}
```

La méthode `ChessPiecesFactory.newPieces()` fabrique une `List` de `Pieces` implémentée dans une `LinkedList`. **Aurait-il été pertinent dans le contexte de ce projet d'utiliser une `ArrayList` ? Justifiez.**

d) Soit la classe `ChessGameGUI` suivante :

```
public class ChessGameGUI extends JFrame implements MouseListener,
                                                     MouseMotionListener, Observer{
   // Attributs …

   public ChessGameGUI(String name, ChessGameControlers chessGameControler, Dimension
                                                                          boardSize) {
          // …
   }
   public void update(Observable arg0, Object arg1) {
          // …
   }
   public void mousePressed(MouseEvent e){
          // …
   }
   public void mouseDragged(MouseEvent e) {
          // …
   }
   public void mouseReleased(MouseEvent e) {
          // …
   }
   public void mouseClicked(MouseEvent e) {     }
   public void mouseMoved(MouseEvent e)   {     }
   public void mouseEntered(MouseEvent e) {     }
   public void mouseExited(MouseEvent e)  {     }
}
```

Pourquoi les méthodes `mouseClicked()` … `mouseExited()` sont-elles écrites alors qu'elles ne font rien ?

2. Maintenance évolutive du projet (10 points)

a) Soit la classe PieceIHM suivante :

```
public  class PieceIHM  implements PieceIHMs {

   Pieces piece;

   public PieceIHM(Pieces piece) {
         this.piece = piece;
   }
   public String toString() {
            return "PieceIHM [name=" + getNamePiece() + ", couleur=" + getCouleur() + ",
               x=" + getX() + ", y=" + getY() + "]";
   }
   public int getX() {
         return piece.getX();
   }
   public int getY() {
         return piece.getY();
   }
   public String getNamePiece() {
         return piece.getName();
   }
   public Couleur getCouleur() {
         return piece.getCouleur();
   }
}
```

A quoi sert cette classe (Expliquez en quoi est-elle pertinente dans cette application) ?

b) On pourrait imaginer ne plus avoir 1 objet `PieceIHM` associé à 1 `Pieces`, (actuellement on a 2 objets instances de `PieceIHMs` pour les `Cavalier` blancs et 2 pour les `Cavalier` noirs), mais 1 objet qui contienne la liste des coordonnées de toutes les pièces de même nom et même couleur, (on aurait alors 1 seul objet `Cavalier` blanc, 1 seul objet `Cavalier` noir). La nouvelle classe serait définie ainsi :

```
public  class PieceIHM implements PieceIHMs {

   String name;        // Ex : Cavalier
   Couleur couleur;    // Ex : Couleur.BLANC
   List<Coord> list;

   PieceIHM(String name, Couleur couleur) {
          this. name = name;
          this.couleur = couleur;
          list = new LinkedList<Coord>();
   }
   public void add(Coord coord){
          list.add(coord);
   }
   // Autres méthodes getXXX() + toString()
}
```

Considérant cette nouvelle version de PieceIHM, écrire la méthode getPiecesIHM() de la classe Jeu qui retourne une List<PieceIHMs>. Pour mémoire, celle utilisant l'ancienne version de la classe `PieceIHM` est rappelée ci-dessous :

```
   public List<PieceIHMs> getPiecesIHM(){
          PieceIHMs newPieceIHM = null;
          List<PieceIHMs> list = new LinkedList<PieceIHMs>();

          for (Pieces piece : pieces){
                if (piece.getX() != -1){ // si la pièce est toujours en jeu
                      newPieceIHM = new PieceIHM(piece);
                      list.add(newPieceIHM);
                }
          }
          return list;
   }

   public List<PieceIHMs> getPiecesIHM(){









   }
```

c) Considérant cette nouvelle version de la méthode `getPiecesIHM()` de la classe `Jeu` qui retourne une `List<PieceIHM>`, écrire la méthode `toString()` de la classe `Echiquier` qui retourne une `String` formatée ainsi :

```
          0     1     2     3     4     5     6     7

0      N_To  N_Ca  N_Fo  N_Re  N_Ro  N_Fo  N_Ca  N_To
1      N_Pi  N_Pi  N_Pi  N_Pi  ____  N_Pi  N_Pi  N_Pi
2      ____  ____  ____  ____  ____  ____  ____  ____
3      ____  ____  ____  ____  N_Pi  ____  ____  ____
4      ____  ____  ____  B_Pi  ____  ____  ____  ____
5      ____  ____  ____  ____  ____  ____  ____  ____
6      B_Pi  B_Pi  B_Pi  ____  B_Pi  B_Pi  B_Pi  B_Pi
7      B_To  B_Ca  B_Fo  B_Re  B_Ro  B_Fo  B_Ca  B_To
```

```
public String toString() {
        String st = "";
        return st;
}
```

Trucs et astuces

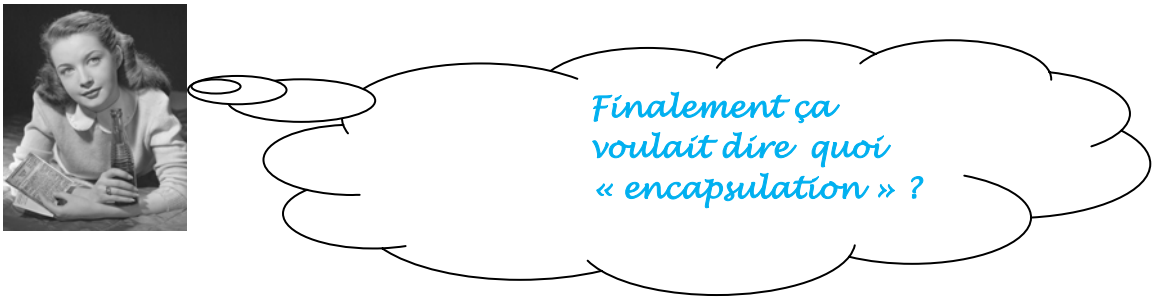
- Extraction d'une sous-chaine :

```
String type = (pieceIHM.getNamePiece()).substring(0, 2);
```

3. Réflexions sur la conception du projet (4 points)

a) A la lumière de la conception du projet (rappel diagramme UML en annexe), **discutez de la pertinence du contrôleur dans ce cas précis.**

b) Soit la pensée suivante :



Expliquez en quoi la conception du projet (Cf. annexe) répond à la question (illustre le concept d'encapsulation).

Annexe : architecture globale de l'application