Correction DS2020_2021

```
public abstract class Piece
{   boolean estBlanc;
    public Piece(boolean estBlanc) {
        this.estBlanc = estBlanc;   }
    public boolean estBlanc() {
        return estBlanc; }
public abstract boolean deplacementValide(Case ori, Case dest);}
```

```
//  Implémentation d'une interface fonctionnelle Dessin avec une seule méthode dessiner()
public interface Dessin {
    void dessiner(); }
// Implémentation de la classe Case avec déclanchement d'une erreur de construction
public class Case {
  private boolean isBlanc;
  private int ligne, colonne ;
  private Piece contenu;
public Case (int ligne, int colonne,boolean isBlanc) throws ErroConst {
 if(this.ligne>8 ||this.colonne>8) throw new ErroConst(this);
   this.ligne=ligne;
   this.colonne=colonne; }
// getters et setters
public Piece getContenu() { return contenu;}
public void setContenu(Piece contenu) { this.contenu = contenu;}
int getLigne(){return ligne;}
int getColonne(){return colonne;}
void setLigne(int ligne){this.ligne=ligne;}
```

```
void setColonne(char colonne){this.colonne=colonne;}

boolean getColor() {return isBlanc;}

void setColor(boolean b){this.isBlanc=b;}
```

// Méthode d'affichage standard qui ne retourne que les attributs de la position

```
public String toString(){

return "ligne= "+ligne+" colonne= "+colonne;}
```

/* **Implémentation de la classe correspondant à l'erreur de construction**. On doit afficher un message d'erreur, ainsi que la position erronée par passage d'objet*/

```
public class ErroConst extends Exception{

   ErroConst(Case ca){

   System.out.println("erreur position");

   System.out.println(ca);    }}
```

// Implémentation de la classe Roi

```
public class Roi  extends Piece implements Dessin{
```

/* attribut de la case courante dans laquelle se trouve le roi*/

```
  private Case courante;

   public Roi(boolean estBlanc) {

       super(estBlanc);}
```

// Implémentation de la méthode dessiner() définie dans l'interface fonctionnelle de Dessin

```
public void dessiner() {

      if (estBlanc) { System.out.println("\u2654");

      } else {     System.out.println("\u265A"); }    }
```

// Implémentation d'une méthode de déplacement du roi héritée

```
 public boolean deplacementValide(Case ori, Case dest) {

      Piece p = dest.getContenu();       int oriligne = ori.getLigne();

      int desligne = dest.getLigne();       int oricolonne = ori.getColonne();

      int descolonne = dest.getColonne();
```

```
if (((oriligne!=desligne)||(oricolonne!=descolonne)) &&  ((Math.abs(desligne - oriligne) <= 1)||
(Math.abs(descolonne- oricolonne) <= 1)) &&((p==null)||(p.estBlanc!=this.estBlanc)))

            return true;

                else return false; }

Case getCase(){return courante;}

void setCase(Case courante) {this.courante=courante;}}

// Implémentation de la classe principale

public class Principal {

/* Méthode statique d'initialisation d'un échiquier qui permet de le retourner sous forme d'un
tableau bidimensionnel*/

  public static Case[][] initialiser() {

   Case[][]echiquier = new Case[8][8];

   try {

   for (int i=0;i<8;i++) {

     for (int j=0;j<8;j++) {

        if(i+j%2==0)     echiquier [i][j]=new Case(i,j,true);

        else echiquier [i][j]=new Case(i,j,false);    } }}

   catch(ErroConst e){   }

   finally{return echiquier;}}

/* Méthode qui permet tester l'interface de Dessin par expression Lambda, et ce en dessinant
un cavalier noir */

public void Tester(){

  dessin d = ()-> System.out.println("\u265E");

     d.dessiner();    }

// Méthode main
```

public static void main(String[] args) {

// Appel d'initialisation de l'échiquier

Case [][] ech = new Case[8][8];

ech = initialiser();

Roi rb = new Roi(true); rb.setCase(ech[4][0]);

Roi rn = new Roi(true); rn.setCase(ech [4][7]);

Scanner sc = new Scanner(System.in);

int i =0; boolean b = true;

int ligne = 0; int colonne = 0;

// Création de deux listes chainées

// liste1 (resp. liste2) pour enregistrer les cases de déplacement de rb (resp. de rn)

List<Case> liste1 = new LinkedList();

List<Case> liste2 = new LinkedList();

// Lancer un jeu de déplacement juste des rois comme suit :

- ✓ afficher la couleur du joueur,
- ✓ puis donner la main au joueur concerné pour qu'il insère les coordonnés de la position de la case de destination via le clavier
- ✓ En cas d'un mouvement valide on enregistre la case dans la liste adéquate, et on effectue la mise à jour de la case courante du roi
- ✓ Idem pour le deuxième joueur
- ✓ Lorsque les deux joueurs terminent une étape du jeu on affiche un message pour pouvoir quitter ou continuer le jeu ; ceci donne la possibilité de sortir de la boucle du jeu à chaque étape */

while (b) {

if (i%2==0){

System.out.println("joueur blanc");

System.out.println("donner les coordonnées deux entiers entre 0 et 7 comme coordonnées de la case destination");

colonne = sc.nextInt();

```
ligne = sc.nextInt();
            if (rb.deplacementValide(rb.getCase(), ech[colonne][ligne])) {
               rb.setCase(ech[colonne][ligne]);
               liste1.add(ech[colonne][ligne]);  }          }
```

else { System.out.println("joueur noir");

System.out.println("donner les coordonnées deux entiers entre 0 et 7 comme coordonnées de la case destination");

```
colonne = sc.nextInt();
ligne = sc.nextInt();
            if (rn.deplacementValide(rn.getCase(), ech[colonne][ligne])) {
               rn.setCase(ech[colonne][ligne]);
               liste2.add(ech[colonne][ligne]);  }   }
```

System.out.println("pour rester taper true ");

System.out.println("pour sortir taper false ");

```
b= sc.nextBoolean();          i=+1;     }
```

// Pour le reste n'utiliser que des streams et la méthode ForEach () pour l'affichage

/* Première manip : en une seule ligne de code, afficher pour chaque case de la liste1 la somme de ses coordonnées*/

```
liste1.stream().map(c->c.getLigne()+c.getColonne()).forEach(System.out::println);
```

/* Deuxième manip : en une seule ligne de code, compléter l'affichage du nombre des cases dans liste1 dont le numéro de ligne est entre 0 et 3. Utiliser mapToInt*/

System.out.println(**liste1.stream().mapToInt(c->c.getLigne()).filter(x->x<4).count()**) ;

/* Troisième manip : en une seule ligne de code, déterminer l'ensemble des cases dans liste2 dont le numéro de colonne est impair*/

Set<Case> ensemble = new HashSet();

ensemble = **liste2.stream().filter(x-> x.getColonne()%2==1). collect(Collectors.toSet()); }}**