

Clement Chesneau 298783.

A.sc.2 CAMPUS STRASBOURG 67400.

# PROJET JAVA

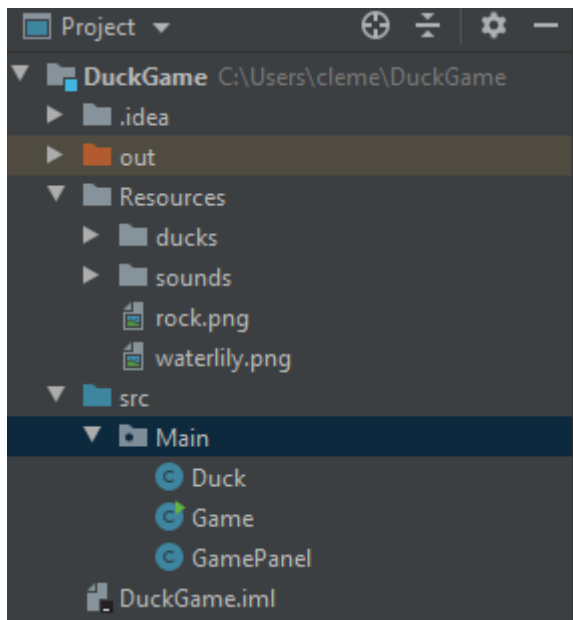
Mon projet contient 3 classes :

La classe Game qui contient la fonction main et ma fenêtre JFrame.

La classe GamePanel qui contient mon JPanel ainsi que ma boucle de jeu.

La classe Duck qui est mon objet Duck.

Toutes ces classes appartiennent au même package Main.

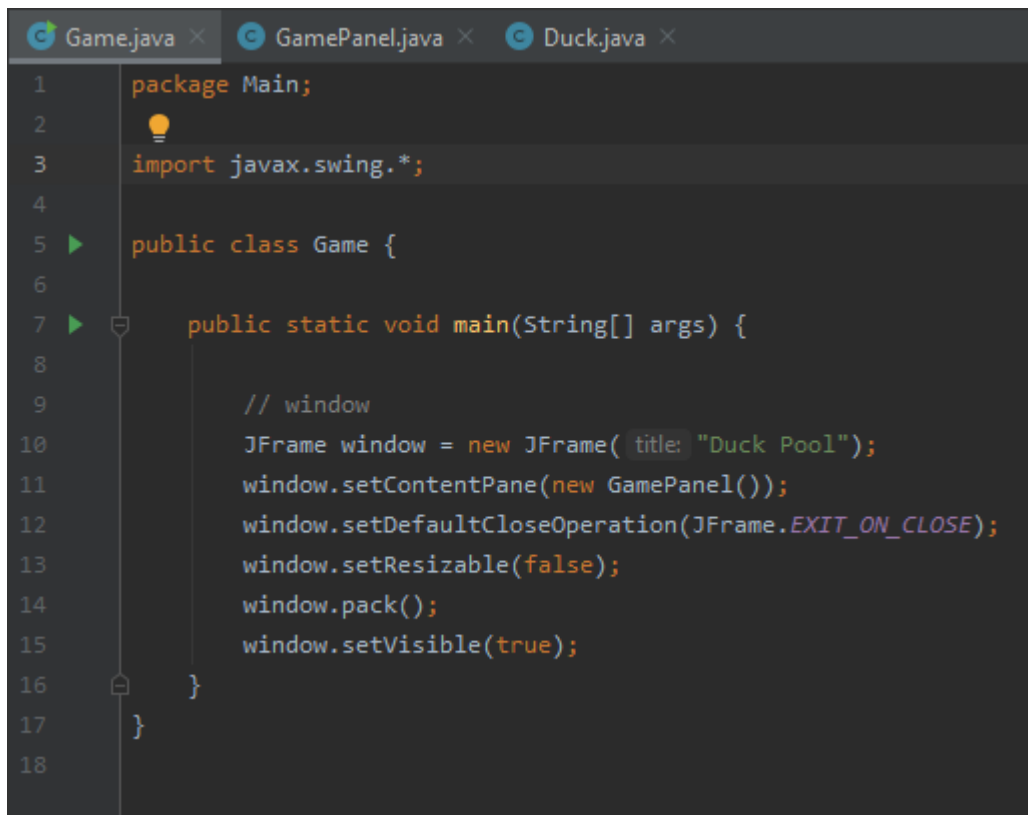


Au début de chaque classe j'importe les packages/variables statiques qui me sont nécessaires :

```
3  import javax.imageio.ImageIO;
4  import java.awt.image.BufferedImage;
5  import java.io.File;
6  import java.io.IOException;
7  import java.util.*;
8
9  import static Main.GamePanel.*;
10
11
12  public class Duck {
```

## Classe Game :

Elle sert uniquement à la création de ma fenêtre de jeu.



```
1 package Main;
2
3 import javax.swing.*;
4
5 public class Game {
6
7     public static void main(String[] args) {
8
9         // window
10        JFrame window = new JFrame( title: "Duck Pool");
11        window.setContentPane(new GamePanel());
12        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13        window.setResizable(false);
14        window.pack();
15        window.setVisible(true);
16    }
17 }
18
```

## Classe GamePanel :

Elle étend JPanel et implémente Runnable pour l'utilisation de la fonction run() étant la fonction principale de mon programme, ainsi que KeyListener pour « écouter » les actions de clique sur une touche, pression sur touche et relâchement d'une touche grâce aux trois fonctions KeyTyped(KeyEvent e), KeyPressed(KeyEvent e) et KeyReleased(KeyEvent e).

La suite ci-dessous...

Dans un premier temps je déclare les variables nécessaires :

```
15
16     private static final long serialVersionUID = 1L;
17     private boolean showMessage = false;
18
19     // fonts
20     Font font = new Font( name: "SansSerif", Font.BOLD, size: 20);
21     Font font1 = new Font( name: "SansSerif", Font.PLAIN, size: 48);
22
23     // dimensions
24     public static final int WIDTH = 533;
25     public static final int HEIGHT = 300;
26     public static final int SCALE = 2;
27
28     // public variables
29     public int[] rocksList = { 0, 0, 500, 10, 1000, 50, 200, 200, 700, 400,
30         | 10, 540, 300, 500, 800, 100, 754, 432, 842, 481};
31     public ArrayList<Duck> ducksList = new ArrayList<Duck>();
32     public ArrayList<int[]> waterlilyList = new ArrayList<int[]>();
33     public static final int TIMEDUCKDIE = 3000;
34
35     // game thread
36     private Thread thread;
37     private boolean running;
38     private int FPS = 60;
39     private long time = 500 / FPS;
40
41     // images
42     private BufferedImage rock;
43     private BufferedImage waterlily;
44
```

La suite ci-dessous...

Ensuite, je configure mon GamePanel, mon thread (que je démarre), mon KeyListener, les images nécessaires puis j'arrive à ma fonction principale.

```
44
45 // panel
46 public GamePanel() {
47     super();
48     setPreferredSize(new Dimension( width: WIDTH * SCALE, height: HEIGHT * SCALE));
49     setFocusable(true);
50     requestFocus();
51     setBackground(new Color( r: 0, g: 128, b: 255));
52 }
53
54 public void addNotify() {
55     super.addNotify();
56     if (thread == null) {
57         thread = new Thread( target: this);
58         addKeyListener( l: this);
59         thread.start();
60     }
61 }
62
63 // initialisation
64 public void init() {
65
66     // run
67     running = true;
68
69     // load images
70     try {
71         rock = ImageIO.read(new File( pathname: "Resources\\rock.png"));
72         waterlily = ImageIO.read(new File( pathname: "Resources\\waterlily.png"));
73     } catch (IOException e) {
74         e.printStackTrace();
75     }
76 }
77
78 // main function
79 public void run() {
80
```

La suite ci-dessous...

Ma fonction run() contient ma boucle de jeu, elle appelle les fonctions nécessaires au fonctionnement de mon programme. J'utilise un minuteur pour effectuer des actions avec un certain délai.

```
79  public void run() {  
80  
81      long timer = 0;  
82      long id = 1;  
83      init();  
84  
85      // game loop  
86      while (running) {  
87
```

```
96          // create new duck  
97          if (timer % 500 == 0) {  
98              Duck duck = new Duck(id, rocksList, ducksList);  
99              ducksList.add(duck);  
100             id++;  
101         }  
102  
103         // create new water lily  
104         waterlilySpawn(timer);  
105  
106         eatWaterLily();  
107  
108         for(Duck duck1: ducksList){  
109             // moves ducks  
110             duck1.move(rocksList, ducksList);  
111             // big ducks whistle  
112             if (duck1.getSmallSize() == 5 && timer % 1500 == 0) {  
113                 audio( path: "Resources\\sounds\\sifflement.wav");  
114             }  
115         }  
116  
117         duckLifeManager();  
118     }
```

On peut voir ci-dessus l'utilisation de mon objet Duck afin de créer un canard dès que mon minuteur / 500 renvoie un nombre entier.

On peut voir aussi différentes fonctions vitales à mon programme, tel que waterlilySpawn(timer) gérant le spawn aléatoire de mes nénuphars en fonction du timer (la fonction ci-dessous).

```

162     public void waterlilySpawn(long timer) {
163         if (timer % 200 == 0) {
164             int randomX = 1 + (int)(Math.random() * ((WIDTH * SCALE - 37) + 1));
165             int randomY = 1 + (int)(Math.random() * ((HEIGHT * SCALE - 37) + 1));
166
167             while (!canGo(randomX, randomY)) {
168                 randomX = 1 + (int)(Math.random() * ((WIDTH * SCALE - 37) + 1));
169                 randomY = 1 + (int)(Math.random() * ((HEIGHT * SCALE - 37) + 1));
170             }
171
172             int[] xAndY = {randomX, randomY};
173             waterlilyList.add(xAndY);
174         }
175     }
176

```

Dans cette fonction on peut voir que je crée deux nombres aléatoires compris dans les dimensions de ma fenêtre correspondant à la position du nénuphar à créer. Tant que ceux-ci ne sont pas acceptés par la fonction canGo qui a pour but de vérifier que les coordonnées ne sont pas utilisées par un caillou, on recommence avec de nouvelles coordonnées.

```

119         repaint();
120         timer++;
121
122         try {
123             Thread.sleep(time);
124         } catch (InterruptedException e) {
125             e.printStackTrace();
126         }
127     }
128

```

Toujours dans ma boucle de jeu, on retrouve une fonction repaint() qui appelle ma fonction paintComponent(Graphics g) étant responsable de l'affichage des différents textes et images sur ma fenêtre. Ainsi qu'un thread.sleep() pour choisir la vitesse de lecture de ma boucle principale. La suite ci-dessous.

```

130 public void paintComponent(Graphics g) {
131     super.paintComponent(g);
132     Graphics2D g2 = (Graphics2D) g;
133
134     // draw rocks
135     for (int i = 0; i < 20; i += 2) {
136         g2.drawImage(rock, rocksList[i], rocksList[i+1], width: 50, height: 50, observer: null);
137     }
138
139     // draw ducks
140     for (Duck duck1: ducksList) {
141         g2.drawImage(duck1.getImage(), duck1.getPos().get(0), duck1.getPos().get(1),
142             duck1.getSize(), duck1.getSize(), observer: null);
143         //g.setFont(font);
144         //g.setColor(Color.WHITE);
145         //g2.drawString(String.valueOf(duck1.getID()), duck1.getPos().get(0)+10, duck1.getPos().get(1)+10);
146     }
147
148     // draw waterlily
149     for (int[] lily: waterlilyList) {
150         g2.drawImage(waterlily, lily[0], lily[1], width: 50, height: 30, observer: null);
151     }

```

On peut voir ici que je parcours 3 listes, chacune permettant au final de me renvoyer les coordonnées de mes cailloux, mes canards (aussi la taille) et mes nénuphars afin de les afficher sur ma fenêtre.

Pour jouer un son ou une musique j'utilise ma fonction audio(path) qui joue le son ce trouvant à la destination « path ».

```

272 // play audio function
273 public void audio(String path) {
274     try {
275         AudioInputStream audioIn = AudioSystem.getAudioInputStream(new File(
276             path));
277         // Get a sound clip resource.
278         Clip clip = AudioSystem.getClip();
279         // Open audio clip and load samples from the audio input stream.
280         clip.open(audioIn);
281         clip.start();
282     } catch (UnsupportedAudioFileException e) {
283         e.printStackTrace();
284     } catch (IOException e) {
285         e.printStackTrace();
286     } catch (LineUnavailableException e) {
287         e.printStackTrace();
288     }
289 }

```

Enfin il y a d'autres fonction sur cette page mais certaines ressemblent à d'autres se trouvant dans ma classe Duck, je les analyse donc plus tard. De plus elles sont plutôt explicites de par leur nom.

## Classe GamePanel :

Elle sert à définir ce qu'est un canard, quel sont ses caractéristiques. On peut ensuite accéder à ses informations grâce à des fonctions que je créer.

### Les variables :

```
14      // variables
15      ArrayList<Integer> pos = new ArrayList<>();
16
17      long id;
18      int size = 1;
19      int timeBeforeDead = TIMEDUCKDIE;
20
21      private BufferedImage duckImage;
22
23      String direction;
24      int whenChange = 0;
25
```

WhenChange étant le temps restant avant que le canard change de direction, les autres sont très explicites.

### Le constructeur :

C'est la fonction que l'on appelle pour créer un nouveau canard.

```
27      // constructor
28      public Duck(long id, int[] rocksList, ArrayList<Duck> ducksList) {
29          ArrayList<Integer> posArray = new ArrayList<Integer>();
30          Collections.addAll(posArray, ...elements: 1 + (int)(Math.random() * ((WIDTH * SCALE - 1) + 1))
31                               , 1 + (int)(Math.random() * ((HEIGHT * SCALE - 1) + 1)));
32
33          while (!canGo(posArray, rocksList, ducksList)) {
34              posArray.set(0, 1 + (int)(Math.random() * ((WIDTH * SCALE - 1) + 1)));
35              posArray.set(1, 1 + (int)(Math.random() * ((HEIGHT * SCALE - 1) + 1)));
36          }
37
38          this.pos.add(posArray.get(0));
39          this.pos.add(posArray.get(1));
40          this.id = id;
41      }
```

On peut voir que cette fonction est similaire à la fonction de spawn des nénuphars, étant donné qu'elle crée une position avec des coordonnées aléatoires en boucle tant que la position n'est pas valable (qu'il y a un rocher à cette endroit). Elle ajoute aussi un id à notre canard.



### Les fonctions set :

Elles permettent de modifier la valeur des variables de notre canard.

```
43 // set functions
44 public void setPos(int x, int y) {
45     this.pos.set(0, x);
46     this.pos.set(1, y);
47 }
48
49 public void setSize(int size) { this.size = size; }
52
53 public void setTimeBeforeDead(int timeBeforeDead) { this.timeBeforeDead = timeBeforeDead; }
56
57 public void setImage() {
```

### Les fonctions get :

Elles permettent de récupérer les variables de notre canard.

```
104 // get functions
105 public ArrayList<Integer> getPos() { return this.pos; }
108
109 public int getSize() {
110     int realSize;
111     if (this.size == 1) {
112         realSize = 15;
113     }
114     else if (this.size == 2){
115         realSize = 20;
116     }
117     else {
118         realSize = 10 * this.size;
119     }
120
121     return realSize;
122 }
123
124 public int getSmallSize() {
125     return this.size;
126 }
127
128 public int getTimeBeforeDead() {
129     return this.timeBeforeDead;
130 }
131
132 public BufferedImage getImage() {
133     return this.duckImage;
134 }
135
136 public long getID() {
137     return this.id;
138 }
```

Enfin, les fonctions restantes permettent à notre canard de se déplacer de manière aléatoire mais un peu intelligemment.

```
// probably essentials functions
public void move(int[] rocksList, ArrayList<Duck> ducksList) {
    // time to change direction
    if (this.whenChange == 0) {
        notSoRandomDirection();
        this.whenChange = 50 + (int)(Math.random() * ((WIDTH * SCALE - 50) + 50));
    }

    // can the duck go that way ?
    while (!canGo(nextPosition(), rocksList, ducksList)) {
        notSoRandomDirection();
    }

    this.setImage();

    switch (this.direction) {
        case "top":
            this.pos.set(1, this.pos.get(1) - 1);
            break;
        case "bottom":
            this.pos.set(1, this.pos.get(1) + 1);
            break;
        case "left":
            this.pos.set(0, this.pos.get(0) - 1);
            break;
        case "right":
            this.pos.set(0, this.pos.get(0) + 1);
            break;
        default:
            System.out.println("We have a problem sir!");
            break;
    }

    whenChange--;
}
```

En effet comme on peut le constater, la fonction move permet de choisir une direction (notSoRandomDirection() et setDirection() s'en charge), en fonction de cette direction de définir la position qu'aurait notre canard si il bougeait (nextPosition()), de recommencer le choix de la position souhaité si elle n'est pas valable (il y a un rocher ou un mur, canGo()). De choisir l'image associée à notre canard en fonction de sa taille (setImage()), puis enfin de changer sa position.

```

176 private void notSoRandomDirection() {
177     if (this.direction == null) {
178         // if no direction, set direction
179         setDirection();
180     }
181     else {
182         String oldDirection = this.direction;
183         // if direction exist, change direction
184         while (this.direction.equals(oldDirection)) {
185             setDirection();
186             // conditions so the duck as less possibilities of going back on his own "steps"
187             if (this.direction.equals("left") && oldDirection.equals("right") ||
188                 this.direction.equals("right") && oldDirection.equals("left")) {
189                 setDirection();
190             }
191             else if (this.direction.equals("top") && oldDirection.equals("bottom") ||
192                     this.direction.equals("bottom") && oldDirection.equals("top")) {
193                 setDirection();
194             }
195         }
196     }
197 }

```

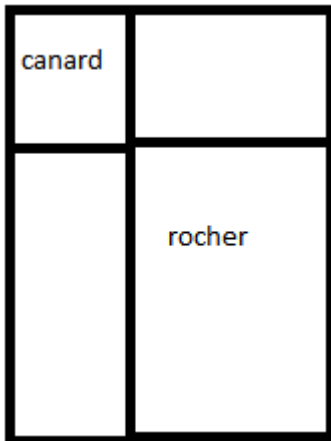
Ici la fonction `notSoRandomDirection()`. Cette fonction permet donc de vérifier si une direction existe, si ce n'est pas le cas, elle en choisit une aléatoirement avec la fonction `setDirection()`, sinon elle en choisit une qui n'est pas la même que la précédente et diminue les chances que la nouvelle direction soit l'inverse de la précédente. Afin d'éviter que le canard revienne presque constamment sur ces propres pas.

```

265 // rock on the way
266 private boolean thereIsARock(int posX, int posY, int[] rocksList) {
267
268     for (int i = 0; i < 20; i+=2) {
269         for (int x = rocksList[i] - this.getSize(); x < 50 + rocksList[i]; x++) {
270             for (int y = rocksList[i+1] - this.getSize(); y < 50 + rocksList[i+1]; y++) {
271                 if (posX == x && posY == y) {
272                     return true;
273                 }
274             }
275         }
276     }
277
278     return false;
279 }
280

```

Ici la fonction `thereIsARock()`. Cette fonction vérifie s'il y a un caillou à la position demandé. Pour cela elle fait une boucle `for()` qui permet de récupérer la position de chaque caillou puis deux autres boucles `for()` pour parcourir tout les points du caillou ainsi que certains points extérieurs n'appartenant pas au caillou car c'est points ne peuvent être utilisées comme coordonnées de notre canard sinon l'image du canard chevaucherait celle du caillou.



Le carré qui entour le canard et le rocher représente la zone qui est définie comme inaccessible pour les coordonnées d'un canard.

FIN