Opérateur = et opérateur ==

Opérateur d'affectation (=) et opérateur de comparaison (==)

1. Opérateur de comparaison
   `if( x = 0)` au lieu de `if(x == 0)`
   Il peut se trouver dans des instructions `if`, `for` et `while`.
2. Opérateur d'affectation
   `int x == 1;` au lieu de `int x = 1;`

Point-virgule mal placé

Vérifiez la présence du point-virgule après les instructions `if` ou les instructions de boucle `for/while`.

Appeler des méthodes avec des arguments incorrects

- Les types de paramètre utilisés pour appeler une méthode doivent correspondre à ceux qui ont servi à la définir.

Conditions limites

- Il est important de tester les **conditions limites**.
- En effet, des erreurs ont tendance à se produire à proximité des valeurs limites d'une variable d'entrée.
- Exemples de condition limite :
  - Données d'entrée (test valide/non valide)
  - Boucles (début et fin des boucles)

Tester les conditions limites pour les boucles

- Cela vous permet de tester des cas limites tels que "inférieur à" et "supérieur à" pour que les conditions d'itération de boucle soient évaluées correctement.
- Par exemple, avec cette boucle :

```
if ( num >= 50 && num <= 100 ) {
//faire quelque chose
}
```

- Pour tester les conditions limites, vous utiliseriez des nombres proches de 50 et 100, c'est-à-dire 49, 50, 51, 99, 100 et 101.

Introduction à JavaFX

Deux méthodes : `start()` et `main()`

- **`start()`** est le point d'entrée de toutes les applications JavaFX.

```
public void start(Stage primaryStage) {
    …
}
```

- **`main()`** reste pourtant nécessaire à vos programmes.

```
public static void main(String[] args) {
    launch(args);
}
```

Boutons = Nœuds

- Certains de ces champs et méthodes sont conçus pour stocker et manipuler des **propriétés visuelles** :
  - `btn.getText()`
  - `btn.setMinHeight()`
  - `btn.setLayoutX()` `//Définir la position x`
  - `btn.setLayoutY()` `//Définir la position y`
  - `btn.isPressed()` `//Clic ?`
- Les objets de ce genre sont appelés **nœuds** JavaFX.

Boutons = Objets

- Les objets `Button` sont comme tous les autres objets.
  - Ils peuvent être instanciés.
  - Ils contiennent des champs.
  - Ils contiennent des méthodes.


```
public void start(Stage primaryStage) {
    Button btn = new Button();
    btn.setText("Say 'Hello World'");
    …
}
```

- Ce que ce code nous apprend :
  - Les objets `Button` contiennent un champ de texte.
  - Les objets `Button` contiennent une méthode permettant de modifier le champ de texte.

Nœuds

- Il existe de nombreux types de nœud JavaFX :

- En général, les objets visuels que vous allez créer :
  - Seront des nœuds
  - Comprendront un nœud comme champ

Interaction des nœuds

- Le code suivant aide à gérer l'interaction des objets Button :

```
public void start(Stage primaryStage) {
    …
    btn.setOnAction(new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            System.out.println("Hello World!");
        }
    });
    …
}
```

- Il s'agit d'une "classe interne anonyme".
  - La syntaxe ne vous semble-t-elle pas désordonnée ?
  - Les **expressions lambda** Java SE 8 représentent une alternative plus élégante.
  - Nous reviendrons sur les expressions lambda un peu plus tard.

Afficher des nœuds

- L'affichage de nœuds se décompose en plusieurs étapes.

```
public void start(Stage primaryStage) {
    Button btn1 = new Button();
    Button btn2 = new Button();
    btn.setText("Say 'Hello World'");
    btn.setText("222");

    StackPane root = new StackPane();
    root.getChildren().add(btn1);
    root.getChildren().add(btn2);
    …
}
```

- Tout d'abord, ajoutez chacun d'eux au **nœud racine**.
  - Il porte généralement le nom `root`.
  - Il s'apparente à une ArrayList contenant l'ensemble des nœuds.

Nœud racine StackPane

- Dans l'exemple donné, le nœud racine est un nœud `StackPane`.

```
StackPane root = new StackPane();
root.getChildren().addAll(btn1, btn2);
```

  - `StackPane` empile les nœuds.
  - Problème : cela pourrait finir par dissimuler des petits boutons et les rendre inaccessibles.

Programmer différents volets en tant que nœuds racine

- Il est facile de concevoir le nœud racine en tant que volet différent.
- Spécifiez simplement un type de référence et un type d'objet différents.

Modifiez ceci… et ceci

```
StackPane root = new StackPane();
root.getChildren().addAll(btn1, btn2);
```

```
TilePane root = new TilePane();
root.getChildren().addAll(btn1, btn2);
```

```
VBox root = new VBox();
root.getChildren().addAll(btn1, btn2);
```

Créer des nœuds

- Les nœuds sont instanciés comme tous les autres objets Java :

```
public void start(Stage primaryStage) {
    Button btn1 = new Button();
    Button btn2 = new Button();
    btn1.setText("Say 'Hello World'");
    btn2.setText("222");
    …
}
```

- Une fois que vous avez instancié un nœud :
  - Il existe et est stocké dans la mémoire allouée.
  - Vous pouvez manipuler ses champs et appeler ses méthodes.
  - Mais il pourrait ne pas s'afficher.

Du moins, pas encore…

Ajouter des nœuds au nœud racine

- Vous pouvez ajouter chaque nœud séparément :

```
root.getChildren().add(btn1);
root.getChildren().add(btn2);
root.getChildren().add(btn3);
```

- Ou en ajouter plusieurs à la fois :

```
root.getChildren().addAll(btn1, btn2, btn3);
```

- En tout cas, n'ajoutez aucun nœud plus d'une fois.
  - Cela générerait une erreur de compilation :

```
root.getChildren().add(btn1);
root.getChildren().add(btn1);
```

Volets en tant que nœuds racine

Chaque **volet** détermine la disposition des nœuds.

Nœud racine Group

- Un nœud `Group` vous permet de placer des nœuds où vous voulez.

```
Group root = new Group();
root.getChildren().addAll(btn1, btn2);
btn1.setLayoutY(100);
```

- Un volet limite vos possibilités en la matière.
  - Il vous empêcherait de les déplacer.
  - Vous ne pourriez pas faire glisser un nœud verrouillé dans un volet.

```
StackPane root = new StackPane();
root.getChildren().addAll(btn1, btn2);
btn1.setLayoutY(100);                    //Sans effet
```

Un nœud Group peut contenir un volet

- Les volets sont également des nœuds.
  - Tout nœud peut être ajouté au nœud racine.
- Un volet peut être utile pour stocker des boutons, des boîtes de dialogue de saisie et autres éléments de GUI.
  - Vous ne pouvez pas vraiment déplacer des nœuds dans un volet.
  - Mais vous pouvez déplacer l'ensemble du volet dans un nœud `Group`. Déplacez le volet comme n'importe quel autre nœud.

Graphe de scène JavaFX

Votre mode d'ajout de nœuds peut être représenté sous la forme d'un **graphe de scène**.

- Le nœud racine contient un volet `HBox`.
- Le volet `HBox` sert de conteneur à boutons.

Objets Scene et Stage

Si vous examinez le reste du programme JavaFX par défaut, vous devriez noter deux autres points :

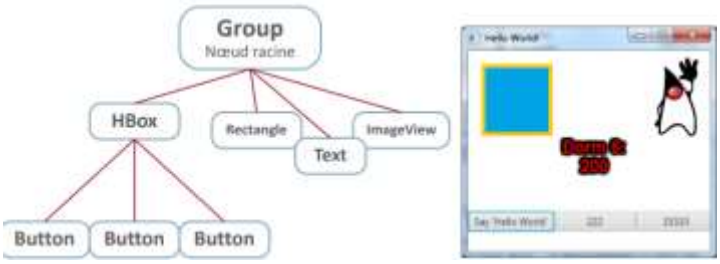
- Un objet Scene (qui contient le nœud racine)
- Un objet Stage (qui contient l'objet Scene)

```
public void start(Stage primaryStage) {
    ...
    Scene scene = new Scene(root, 300, 250);

    primaryStage.setTitle("Hello World!");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Graphe de scène

- Le volet `HBox` assure l'organisation et le bon emplacement de la GUI.
- Le reste de la fenêtre peut être utilisé pour d'autres nœuds.



Qu'est-ce que l'objet Scene ?

Un objet `Scene` se caractérise par certaines propriétés :

- Graphe de scène
  - L'objet `Scene` est le conteneur réunissant tous les composants du graphe de scène JavaFX.
- Taille
  - Vous pouvez régler la largeur et la hauteur de l'objet `Scene`.
- Arrière-plan
  - L'arrière-plan peut être défini dans `Color` ou BackgroundImage.
- Propriétés du curseur
  - L'objet `Scene` peut détecter les événements associés à la souris et gère les propriétés du curseur.

```
Scene scene = new Scene(root, 300, 250, Color.BLACK);
```

Nœud racine, Largeur, taille, Arrière-plan

Qu'est-ce que l'objet Stage ?

Considérez l'objet Stage comme la fenêtre de l'application.

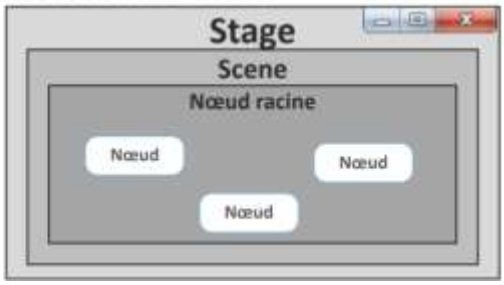
Voici deux de ses propriétés majeures :

- Titre
  - Vous pouvez définir le titre de l'objet Stage.
- Scene
  - L'objet Stage contient un objet `Scene`.

```
primaryStage.setTitle("Hello World!");
primaryStage.setScene(scene);
primaryStage.show();
```

Hiérarchie - Animation

- L'objet Stage est le conteneur de plus haut niveau.
- L'objet Stage contient un objet Scene.
- L'objet Scene contient un nœud racine.
- Le nœud racine contient d'autres nœuds.

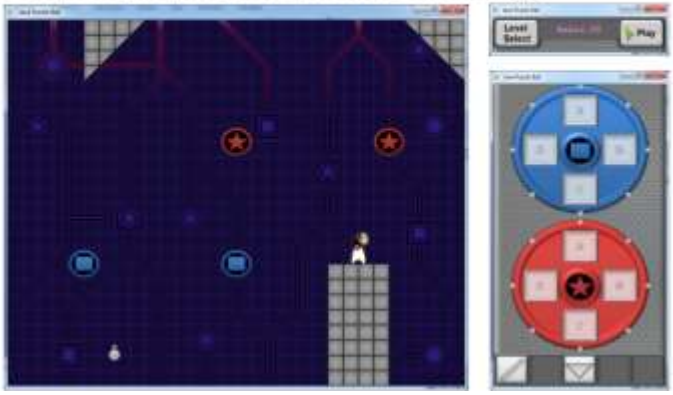


Plusieurs objets Scène pour un seul objet Stage

Vous pouvez échanger les objets Scene dans un objet Stage.

Plusieurs objets Scene pour plusieurs objets Stage

Vous pouvez également créer plusieurs objets Stage.



Que faire avec des couleurs dans JavaFX ?

- Colorier des formes

- Créer des dégradés

- Colorer des images

Couleurs et formes

JavaFX et sa classe Color

- Vous pouvez stocker des couleurs en tant que variables

```
Color color = Color.BLUE;
```

- Vous pouvez transmettre des couleurs dans des méthodes :

```
Scene scene = new Scene(root, 300, 250, Color.BLACK);
```

  – Le code présenté donne à l'objet Scene un arrière-plan noir.

- Mais avant d'utiliser une couleur...

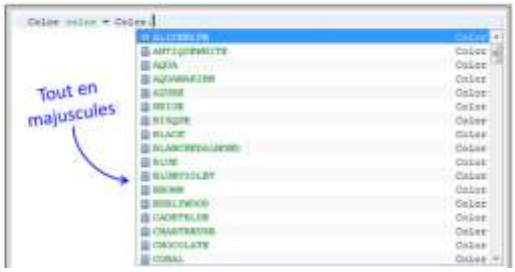
```
import javafx.scene.paint.Color;
```

  – Vous devez d'abord effectuer l'import suivant :

  – Ignorez les autres suggestions d'import de couleur NetBeans.

Référencer une couleur

- JavaFX propose de nombreuses couleurs.
- Entrez `Color.` dans NetBeans pour connaître la liste complète des couleurs disponibles.



Personnaliser une couleur

- Si vous n'êtes pas satisfait des couleurs de JavaFX, vous avez la possibilité de créer les vôtres.
- La classe Color contient les méthodes requises :

  – Créez une couleur personnalisée en mélangeant les composants rouge, vert et bleu.

  – Vous pouvez aussi en régler l'opacité.

Composant	Plage de valeurs
Rouge	0-255
Vert	0-255
Bleu	0-255
Opacité	0,0-1,0

Règles de la synthèse additive de couleurs

Exemples :

Code	Couleur	
`Color.rgb(255, 0, 0);`	Rouge	100 % rouge
`Color.rgb(0, 255, 0);`	Vert	100 % vert
`Color.rgb(0, 0, 255);`	Bleu	100 % bleu
`Color.rgb(255, 255, 0);`	Jaune	Pas de bleu
`Color.rgb(0, 0, 0);`	Noir	Pas de couleur
`Color.rgb(255, 255, 255);`	Blanc	Toutes les couleurs à 100 %

ORACLE

Ceci est un rectangle

- Voici comment instancier un rectangle JavaFX :

```
Rectangle rect = new Rectangle(20, 20, 100, 200);
```

Position X, Position Y, Largeur, taille

- Vous devez d'abord effectuer l'import suivant :

```
import javafx.scene.shape.Rectangle;
```

  – Ignorez les autres suggestions d'import de rectangle NetBeans.

Méthodes importantes pour les rectangles

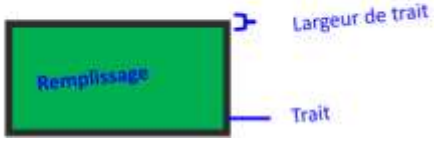
- Le constructeur et les méthodes ci-dessous donnent une idée des propriétés d'un rectangle :
  - `setX(double d)`
  - `setY(double d)`
  - `setWidth(double d)`
  - `setHeight(double d)`
  - `setFill(Paint paint)`
  - `setStroke(Paint paint)`
  - `setStrokeWidth(double d)`

  Celles-ci acceptent une couleur comme argument.

(Il existe bien d'autres méthodes Rectangle que ces sept-là.)

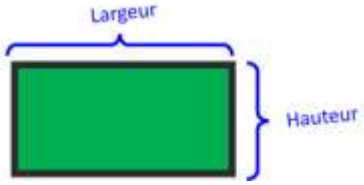
Descriptions des méthodes - Partie 1

- **`setFill(Paint paint)`**
  - Permet de définir la couleur du rectangle
- **`setStroke(Paint paint)`**
  - Permet de définir la couleur de la bordure du rectangle
- **`setStrokeWidth(double d)`**
  - Permet de définir la largeur de la bordure du rectangle



Descriptions des méthodes - Partie 2

- **`setX(double d)`**
- **`setY(double d)`**
  - Permet de définir la position X ou Y du rectangle
- **`setWidth(double d)`**
- **`setHeight(double d)`**
  - Permettent de définir la largeur/hauteur du rectangle



Modifier la position d'un nœud

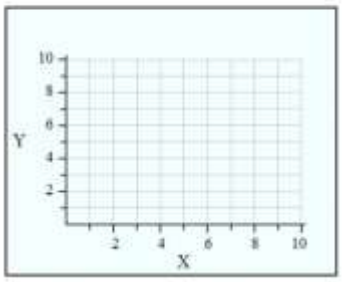
- Nous avons vu quelques façons de modifier la position d'un nœud... mais laquelle est préférable ?
- **`setX(double d)`**
- **`setY(double d)`**
  - Celles-ci sont préférables dans la plupart des cas.
- **`setLayoutX(double d)`**
- **`setLayoutY(double d)`**

  setX() ne fonctionnera pas dans ce cas.
  - Utilisez celles-ci si votre nœud est verrouillé dans un volet de disposition tel que `FlowPane`.
  - Ou si `setX()` est indisponible, ce qui est le cas avec des éléments de l'interface utilisateur tels que les boutons.

Positionner un nœud

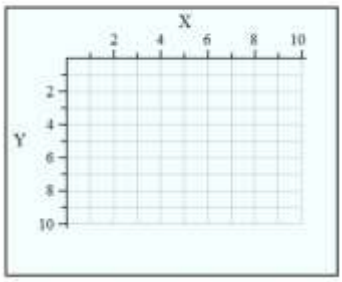
- La plupart des nœuds sont placés en fonction de leur angle supérieur gauche.
  - Et non par rapport à leur centre physique.
- Si vous appelez `setX(100)` sur un nœud...
  - La position X de l'angle supérieur gauche du nœud est fixée à 100.

Systèmes de coordonnées



Système de coordonnées mathématiques

- L'origine se trouve dans l'angle inférieur gauche.



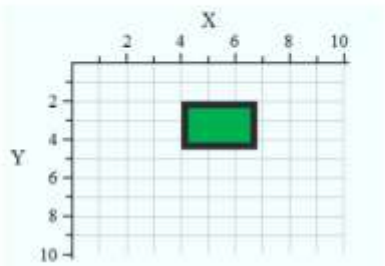
Système de coordonnées JavaFX

- L'origine se trouve dans l'angle supérieur gauche.
- L'axe des ordonnées est inversé.

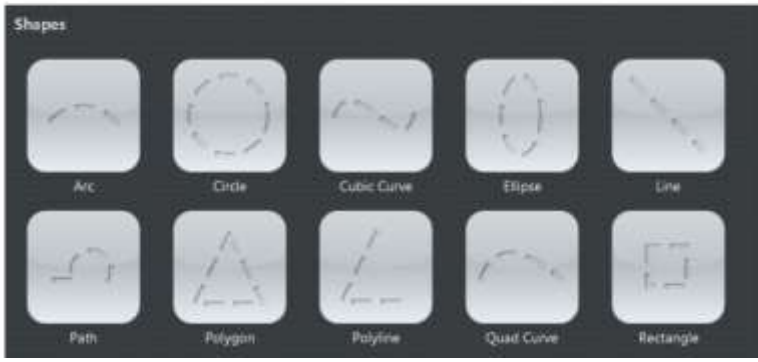
Exemple de positionnement

Méthodes appelées pour positionner le rectangle aux coordonnées (4,2) :

- `setX(4);`
- `setY(2);`



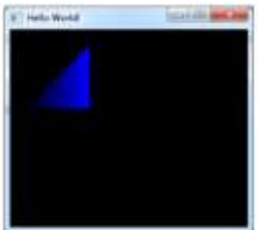
JavaFX et ses nombreuses formes



JavaFX Ensemble

- Cet outil contient des exemples de code des fonctionnalités JavaFX.
- Nous l'avons souvent consulté lors du développement de Java Puzzle Ball.
- C'est un outil utile pour découvrir JavaFX et résoudre les problèmes rencontrés.

- Explorez JavaFX Ensemble.
- Trouverez-vous comment créer un triangle rectangle avec un dégradé de couleur ?



Explorer l'outil Ensemble : Exemple de dégradé linéaire

- Ce que montre l'exemple de dégradé linéaire :
  - Comment créer un dégradé :

```
//Créer un dégradé linéaire simple
LinearGradient gradient1 = new LinearGradient(0, 0, 1, 0, true,
CycleMethod.NO_CYCLE, new Stop[] {
        new Stop(0, Color.DODGERBLUE),
        new Stop(1, Color.BLACK)
});
```

  - Comment colorier une forme avec un dégradé :

```
//Premier rectangle
Rectangle rect1 = new Rectangle(0,0,80,80);

//Définir le remplissage du rectangle
rect1.setFill(gradient1);
```

  - N'oubliez pas d'effectuer les imports adéquats.

Explorer l'outil Ensemble : Exemple de polygone

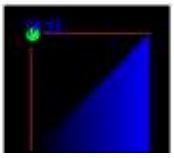
- Ce que montre l'exemple de polygone :
  - Comment créer un polygone à partir d'un tableau de points :

```
//Triangle simple
Polygon polygon1 = new Polygon(new double[]{
        45 , 10 ,
        10 , 80 ,
        80 , 80 ,
});
```

- Combinez ce code à l'exemple de dégradé et vous tiendrez votre solution.
  - Mieux encore, vous comprendrez à quel point l'outil Ensemble constitue une ressource précieuse.
  - Il pourrait être très utile au moment de résoudre le problème pratique.

Polygone

- Les méthodes d'un polygone sont semblables à celles d'un rectangle.
  - Les nœuds partagent les mêmes méthodes.
- Faites un essai avec `setLayoutX()`.
  - Vous noterez que le polygone est placé en fonction de son angle supérieur gauche.



Quelques secrets de Java Puzzle Ball

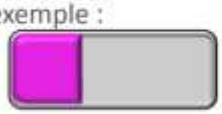
- Nous avons dessiné des lignes et des polygones pour détecter les collisions.
  - Ces lignes sont masquées dans la dernière version.

- Nous avons également dessiné deux octogones autour de chaque bumper.
  - L'octogone interne gère la détection des collisions.
  - L'octogone externe détermine si la balle est assez loin pour que le bumper puisse tourner.
- Nous avons dû fournir des efforts supplémentaires afin de placer et de faire pivoter les nœuds comme nous le voulions.

Graphiques, sons et événements associés à la souris

Utiliser vos propres graphiques

- JavaFX fournit des formes, du texte et des éléments pour l'interface utilisateur.
  - Cependant, si vous êtes artiste dans l'âme, vous pouvez utiliser des graphiques de votre composition.
- Par exemple :



  - L'habillage du bouton servant à sélectionner le niveau ne provient pas de JavaFX.
  - En revanche, nous avons utilisé JavaFX pour y ajouter le numéro des niveaux, le texte correspondant et l'image de Duke.

Pourquoi combiner image et ImageView ?

- L'avantage principal est l'**animation**.
  - Vous pouvez échanger des images à volonté dans ImageView
- Le ventilateur de Java Puzzle Ball en est un exemple.
  - Lorsqu'il souffle, il va et vient entre deux images.

- Les boutons personnalisés en profitent également.
  - Vous pouvez utiliser des images différentes pour les boutons en fonction de leur état :
    - Le curseur passe-t-il sur le bouton ?
    - L'utilisateur clique-t-il sur le bouton ?

Créer des objets avec des propriétés de nœud

- Jusqu'à présent, nous avons écrit l'ensemble du code JavaFX dans la méthode `start()`.
  - Cela ressemble au début du cours, lorsque la majorité du code était écrite dans la méthode `main()`.
- Or, ce n'est pas ainsi que le code orienté objet doit être écrit.
  - Les objets devraient plutôt comprendre des champs de nœud.
- Les méthodes `start()` et `main()` sont destinées à être des pilotes.

Emplacement des fichiers

- Assurez-vous que les fichiers se trouvent au bon emplacement.

```
Image image = new Image(getClass().getResource("Images/Duke.png"));
```

- `Images/Duke.png` fait référence à un sous-dossier du dossier `GoalTest`.
  - *...\GoalTest\src\goaltest\Images*

Dossier du projet — Source — Package principal — Autre package

  - Ou un package compris dans un autre package

Image JavaFX et **ImageView**

- Une **image** est un objet qui décrit l'emplacement d'un fichier graphique (`.png`, `.jpg`, `.gif`...).

```
Image image;
String imagePath = "Images/Fan1.png";
image = new Image(getClass().getResource(imagePath).toString);
```

- **ImageView** représente le nœud réel.
  - L'appel de son constructeur nécessite un argument Image.

```
ImageView imageView = new ImageView(image);
```

  - Par ailleurs, ImageView contient les mêmes propriétés que les autres nœuds : position X, position Y, largeur, hauteur...

Conseils ImageView

- Comment créer des images :

```
Image image1= new Image(getClass().getResource("Images/fan1.png").toString());
Image image2= new Image(getClass().getResource("Images/fan2.png").toString());
```

- Comment créer ImageView :

```
ImageView imageView = new ImageView(image1);
```

- Comment échanger une image dans ImageView :

```
imageView.setImage(image2);
```

  - `imageView` conserve ses propriétés, dont le placement.

N'oubliez pas d'importer
`javafx.scene.image.Image;` et
`javafx.scene.image.ImageView;`

Exemple : Classe Goal

- Champs
  - `private Image dukeImage;`
  - `private ImageView dukeImageView;`

- Constructeur
  - Il accepte des arguments pour les positions `x` et `y`.
  - Il affecte l'image à l'objet ImageView correspondant.
  - Il place `dukeImageView` en fonction des arguments `x` et `y`.

Mettre un nœud à l'échelle

- Il est très facile d'élargir un rectangle :

- En revanche, avec un objet ImageView...
  - Le résultat peut être affreux !

Mettre un nœud à l'échelle correctement

- JavaFX est très performant pour la mise à l'échelle des graphiques.
  - La qualité de l'image est moins susceptible de se détériorer.
- Vous avez la possibilité de conserver les proportions des objets ImageView.
  - La largeur et la hauteur sont ajustées simultanément.

```
imageView.setPreserveRatio(true);
imageView.setFitWidth(25);
```

Définir l'ordre des nœuds correctement

- L'ordre d'ajout des nœuds au nœud racine détermine leur ordre d'affichage.
- Les nœuds ajoutés en premier se retrouvent sous les suivants.

```
root.getChildren().addAll(gloveImageView, dukeImageView);
```

- Comment corriger cela ?
  - Modifiez l'ordre dans lequel vous ajoutez les nœuds au nœud racine.
  - Appliquez ImageView au-dessus ou en dessous.

```
gloveImageView.toFront();      //L'un ou l'autre
dukeImageView.toBack();        //résoudra le problème
```

Définir l'ordre des nœuds

- Les testeurs de Java Puzzle Ball n'ont pas toujours compris qu'ils étaient censés envoyer la balle à Duke.
- Pour résoudre ce problème, nous lui avons ajouté un gant.
- Duke et le gant sont deux objets ImageView distincts.
  - Il convenait de bien les ordonner de sorte que le gant ne soit pas dissimulé par la main.

Correct

Incorrect

Similitudes entre image et son

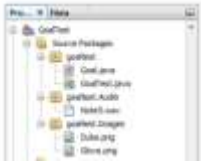
- Création d'un objet Image JavaFX :

```
Image image = new Image(getClass().getResource("Images/fan1.png").toString());
```

- Elle ressemble à celle d'un objet **Audio** JavaFX.

```
Audio audio = new Audio(getClass().getResource("Audio/Note5.wav").toString());
```

- Il est courant de stocker des images et des sons dans des packages/dossiers distincts.



Différences entre image et son

- Un objet Audio décrit l'emplacement d'un fichier audio (`.wav`, `.mp3`…).

```
Audio audio = new Audio(getClass().getResource("Audio/Note5.wav").toString());
```

- Par ailleurs, contrairement aux images :
  - Il n'existe aucun équivalent audio d'ImageView.
  - Les contenus audio peuvent être lus en référençant l'objet Audio directement.

```
audio.play();
```

  - Vous pouvez appeler de nombreuses autres méthodes Audio.

Evénements de souris et de clavier

- Les nœuds sont capables de détecter les événements de souris et de clavier.
  - Les objets ImageView aussi d'ailleurs !
  - Vous n'êtes pas limité aux boutons et autres composants GUI.
- Quelques méthodes de détection utiles :
  - `setOnMouseClicked()`
  - `setOnMouseDragged()`
  - `setOnMouseEntered()`
  - `setOnMouseExited()`
  - `setOnMouseMoved()`
  - `setOnMousePressed()`
  - `setOnMouseReleased()`

N'oubliez pas d'importer `javafx.scene.input.MouseEvent`

Expressions lambda

- Ces méthodes utilisent un argument spécial, appelé **expression lambda** :

```
imageView.setOnMousePressed( /*Expression lambda*/ );
```

- Les expressions lambda emploient une syntaxe particulière :

Pas de point-virgule

```
(MouseEvent me) -> System.out.println("Clic")
```

- Grâce aux accolades, les expressions lambda peuvent contenir plusieurs instructions :

```
(MouseEvent me) -> {
        System.out.println("Instruction 1");
        System.out.println("Instruction 2");
}
```

Points-virgules

Expressions lambda en tant qu'arguments

- Ces deux éléments combinés donnent le résultat suivant :

```
imageView.setOnMousePressed( (MouseEvent me) -> {
        System.out.println("Instruction 1");
        System.out.println("Instruction 2");
} );
```

- Ce que fait ce code :
  - Il permet à l'objet `imageView` de détecter un clic de souris à tout moment.
  - Si cela se produit, les deux instructions print sont exécutées.
  - Sinon, ce code est ignoré.

Objet MouseEvent

- Un objet MouseEvent existe uniquement dans la portée de l'expression lambda.
- Il contient de nombreuses propriétés et méthodes utiles :

```
imageView.setOnMousePressed( (MouseEvent me) -> {
        System.out.println(me.getSceneX());
        System.out.println(me.getSceneY());
} );
```

- Dans l'exemple donné :
  - `me` est l'objet MouseEvent.
  - Le code accède à `me` pour afficher les positions X et Y du curseur lors d'un clic sur `imageView`.

Ecoute des événements

- Lorsque vous écrivez du code pour les objets MouseEvent :
  - Vous demandez à un nœud d'écouter un événement particulier.
  - Toutefois, il n'est pas nécessaire que les événements se produisent.
- En effet, tant qu'un nœud écoute :
  - Il peut détecter tous les événements, à tout moment.
- Il peut écouter plusieurs événements.

```
imageView.setOnMousePressed( /*Expression lambda*/ );
imageView.setOnMouseDragged( /*Expression lambda*/ );
imageView.setOnMouseReleased(/*Expression lambda*/ );
```

Méthodes MouseEvent

- **`getSceneX()`**
- **`getSceneY()`**
  - Renvoie une valeur `double`.
  - Renvoie la position du curseur dans l'objet Scene JavaFX.
  - L'angle supérieur gauche de l'objet Scene représente la position (0,0).

- **`getScreenX()`**
- **`getScreenY()`**
  - Renvoie une valeur `double`.
  - Renvoie la position du curseur sur l'écran de votre ordinateur.
  - L'angle supérieur gauche de l'écran de votre ordinateur correspond aux coordonnées (0,0).