Programmation Orientée Objet Java

Slick

INTRODUCTION

Slick 2D

- Bibliothèque de composants Java dédiée à la conception de jeux.
- Site officiel : `http://slick.ninjacave.com`
- Repose sur une autre bibliothèque : LWJGL.
- Slick et LWJGL sont des bibliothèques libres, licence BSD.

Installation des fichiers

- En salle de TP : installés dans `C:\dev\java\slick`
- Sur une machine personnelle : copier le dossier `slick` à un emplacement facile d'accès.

Attention aux changements machine personnelle / pc de l'école

Les projets Java sont configurés à partir du chemin d'installation de Slick.
Il faut vérifier/changer la configuration du projet à chaque fois.

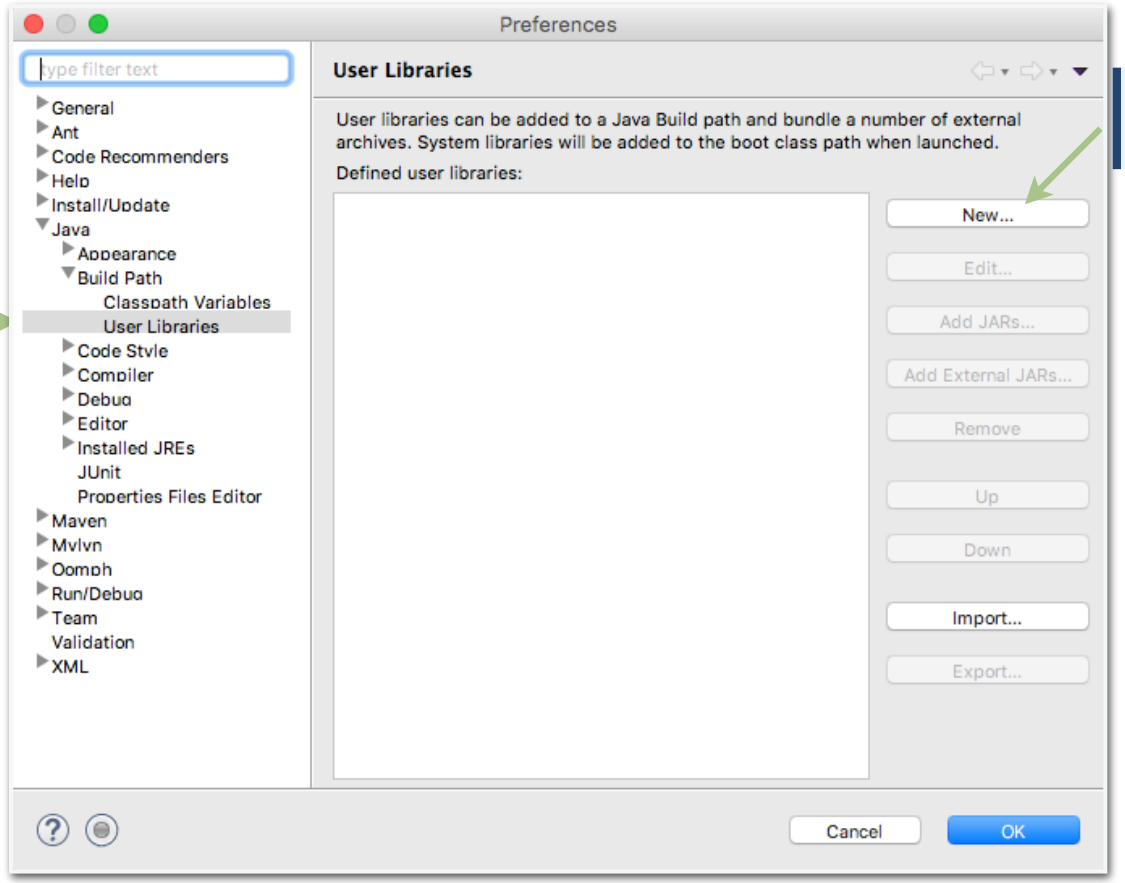
CRÉATION D'UN PROJET AVEC SLICK

- La check-list :
  - Création d'un projet Java
  - Intégration de `slick.jar` et `lwjgl.jar`
  - Intégration du chemin des bibliothèques natives.
  - Création de la classe d'état.
  - Création de la classe du jeu.
- Fichiers .jar & natives
  - JAR : archives de classes Java pré-compilées.
  - Mode de distribution classique de bibliothèques en Java.
  - Native : bibliothèque conçue pour un système d'exploitation précis.
  - Pour LWJGL : afin de dialoguer avec la carte graphique et les périphériques d'entrée (clavier, souris, joystick).

CRÉER UNE BIBLIOTHÈQUE 1/7

- À faire avant de créer un projet
  - Nécessaire une seule fois par workspace
  - Utilisable pour plusieurs projets (dans le même workspace)
- Création d'une bibliothèque
  - Ouvrir les préférences d'Eclipse
  - Déplier *Java* > *Build Path* > *User Libraries*

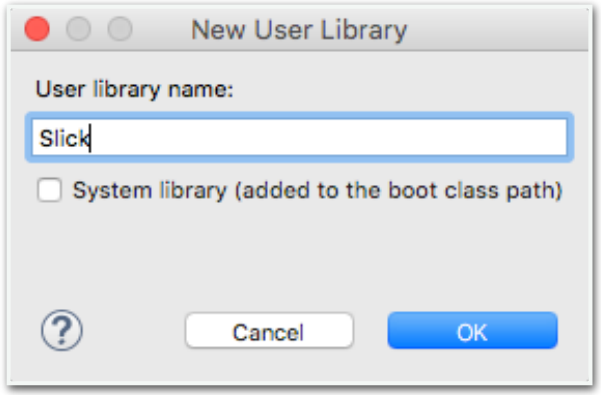
User Libraries



Cliquer sur New...

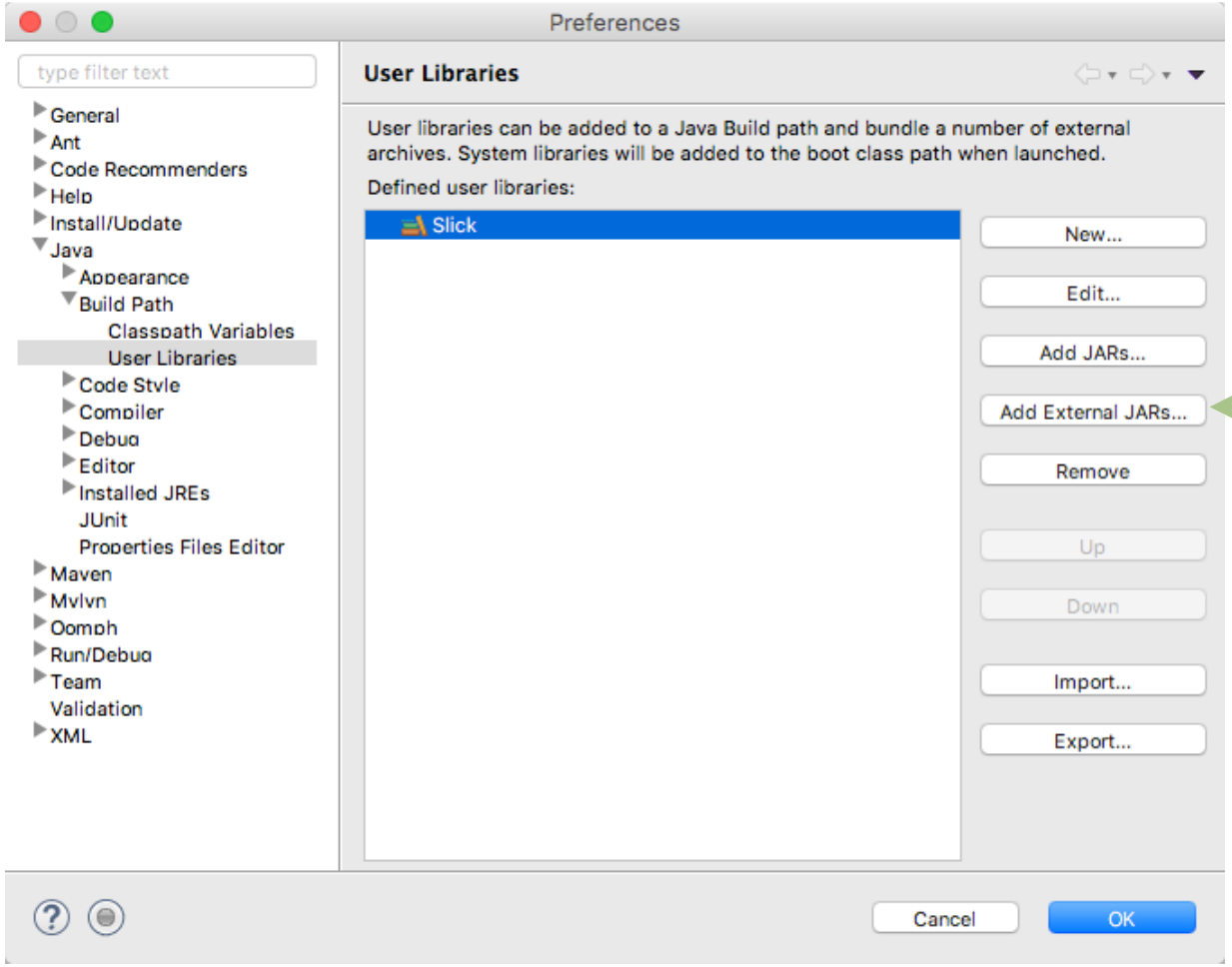
CRÉER UNE BIBLIOTHÈQUE 2/7

- Création de la bibliothèque
  - Apparition d'une boîte de dialogue pour saisir le nom de la bibliothèque.
  - Libre choix du nom.
  - Au plus simple : entrer *Slick*



CRÉER UNE BIBLIOTHÈQUE 3/7

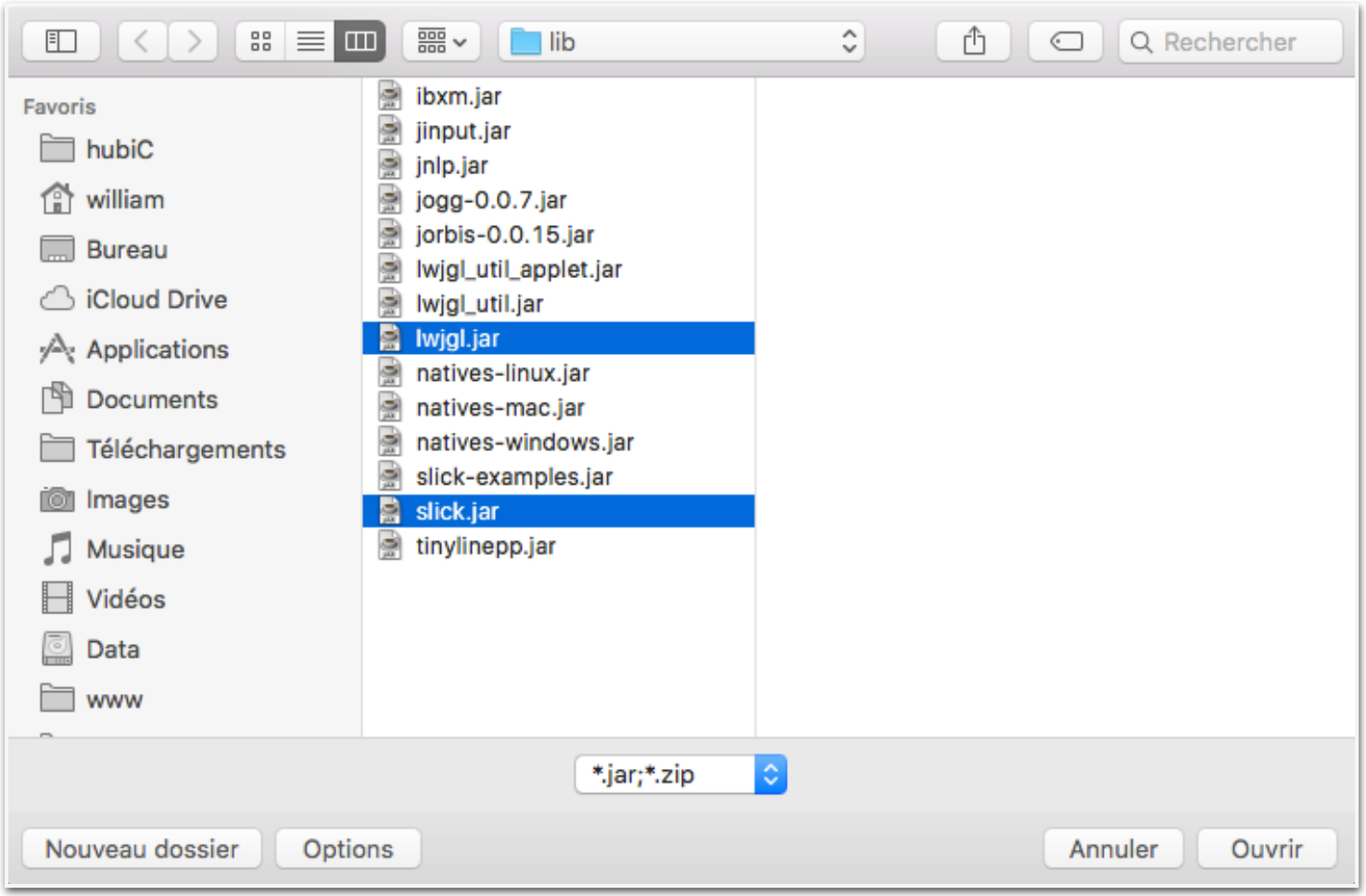
- Intégration des fichiers .jar
  - Sélectionner la bibliothèque dans la liste
  - Cliquer sur *Add External JARs...*



Add External JARs...

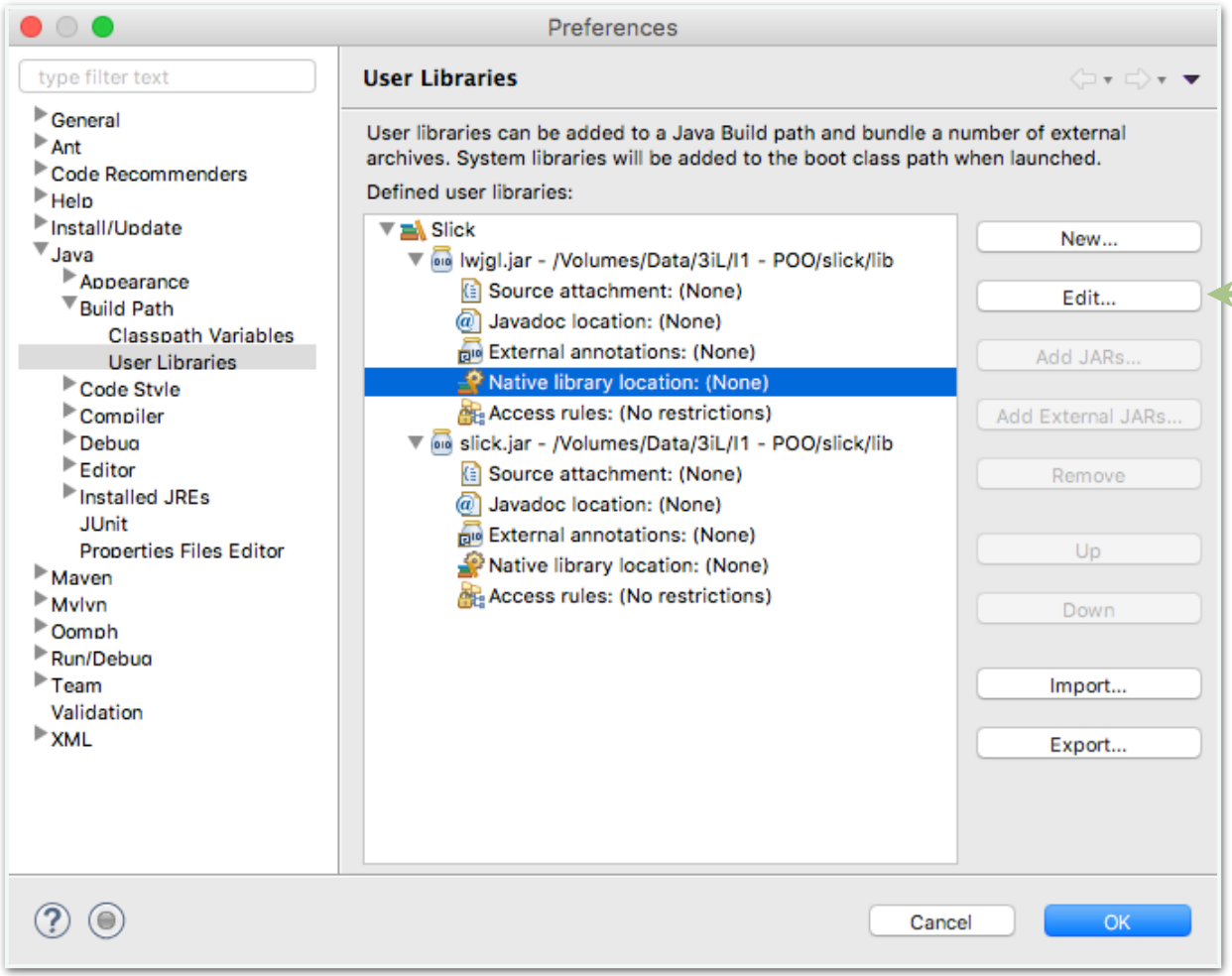
CRÉER UNE BIBLIOTHÈQUE 4/7

- Intégration des fichiers .jar
  - Se rendre dans le dossier *lib* de *slick*
  - Sélectionner *lwjgl.jar* et *slick.jar*



CRÉER UNE BIBLIOTHÈQUE 5/7

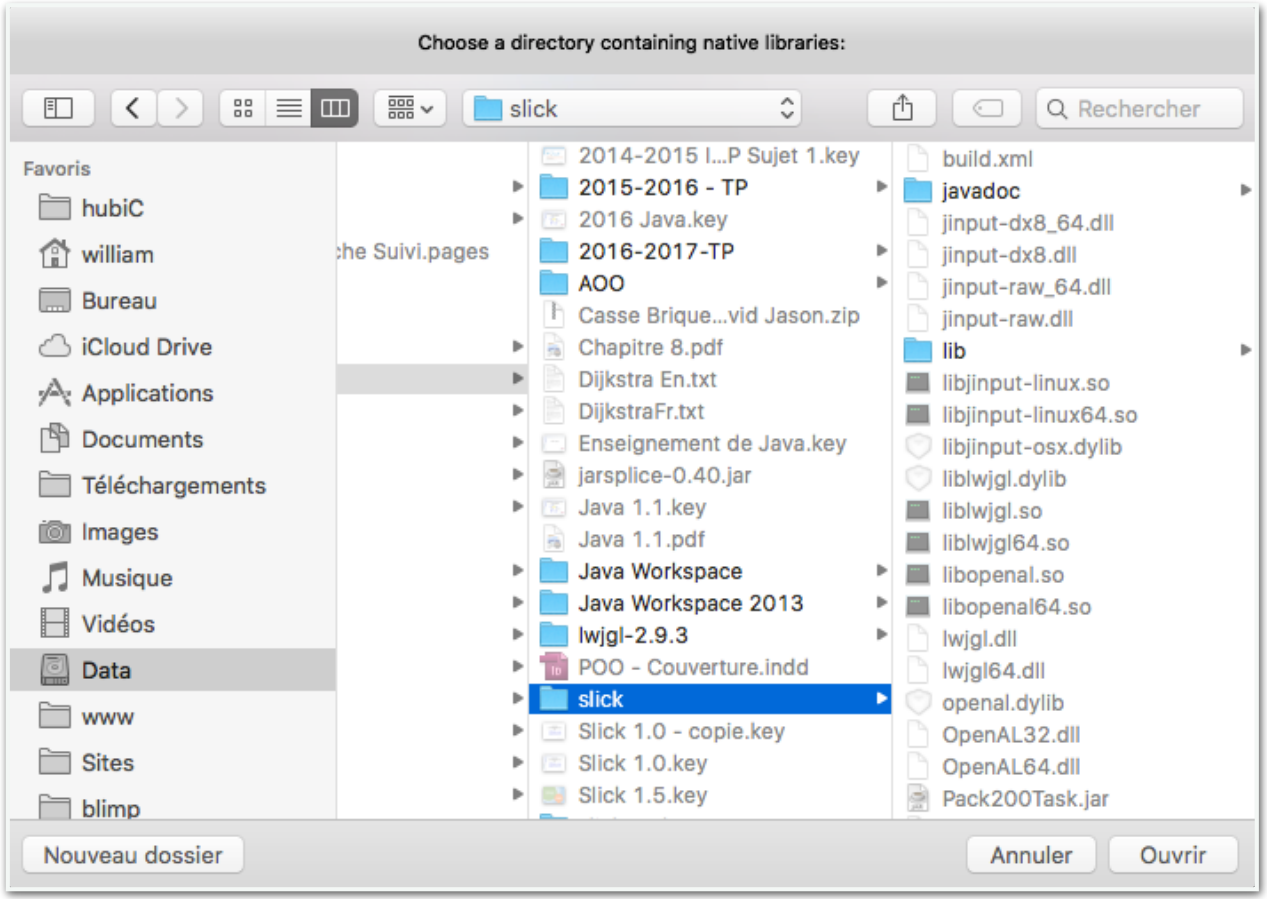
- Intégration des natives
  - Natives : bibliothèques adaptées au système d'exploitation
  - Sélectionner *Native library* dans *lwjgl.jar*
  - Appuyer sur *Edit...*



Edit...

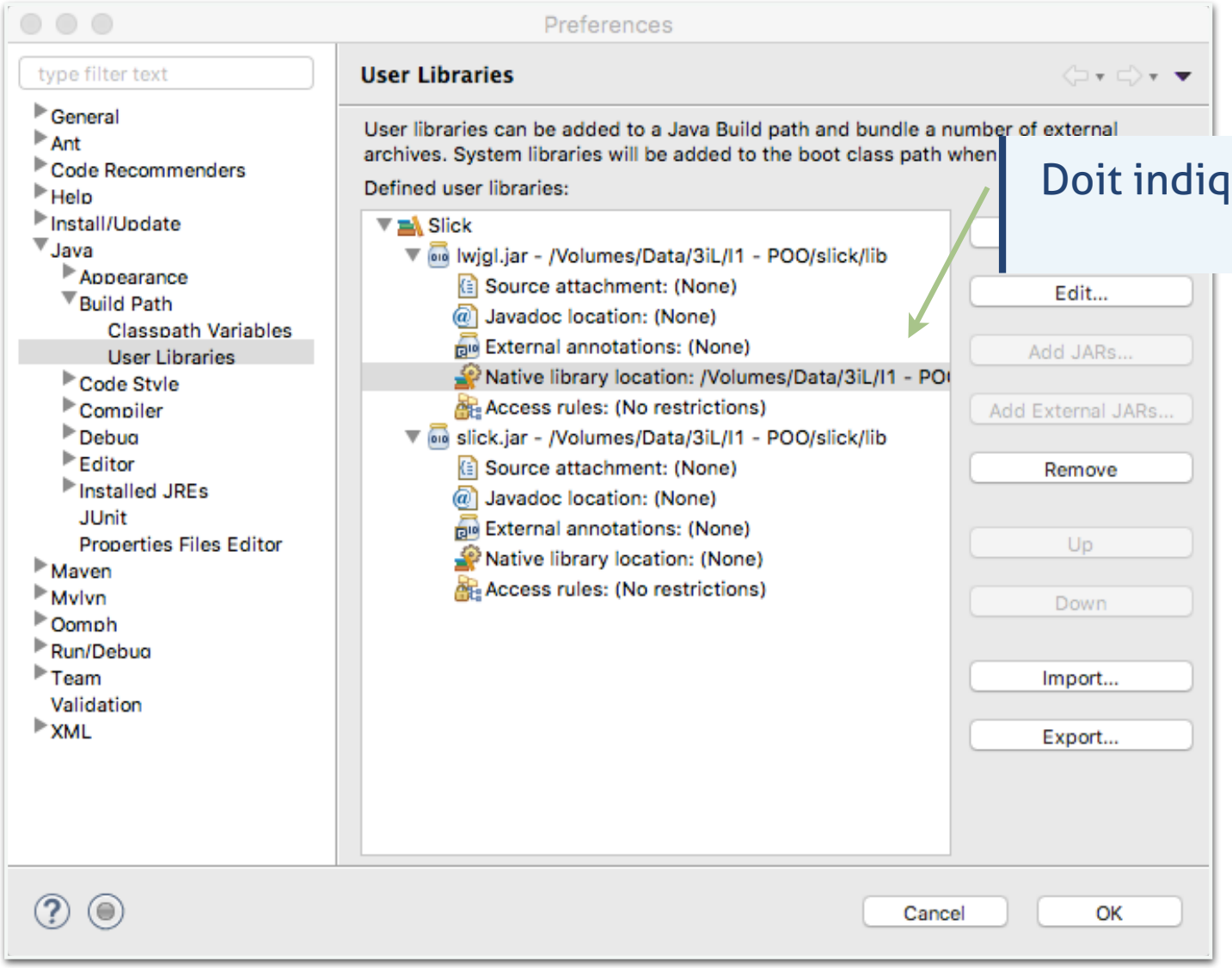
CRÉER UNE BIBLIOTHÈQUE 6/7

- Choisir le dossier Slick
  - Cliquer sur *External Folder...*
  - Sélectionner le dossier *slick* lui-même (dossier parent du dossier lib)



CRÉER UNE BIBLIOTHÈQUE 7/7

- Fermer



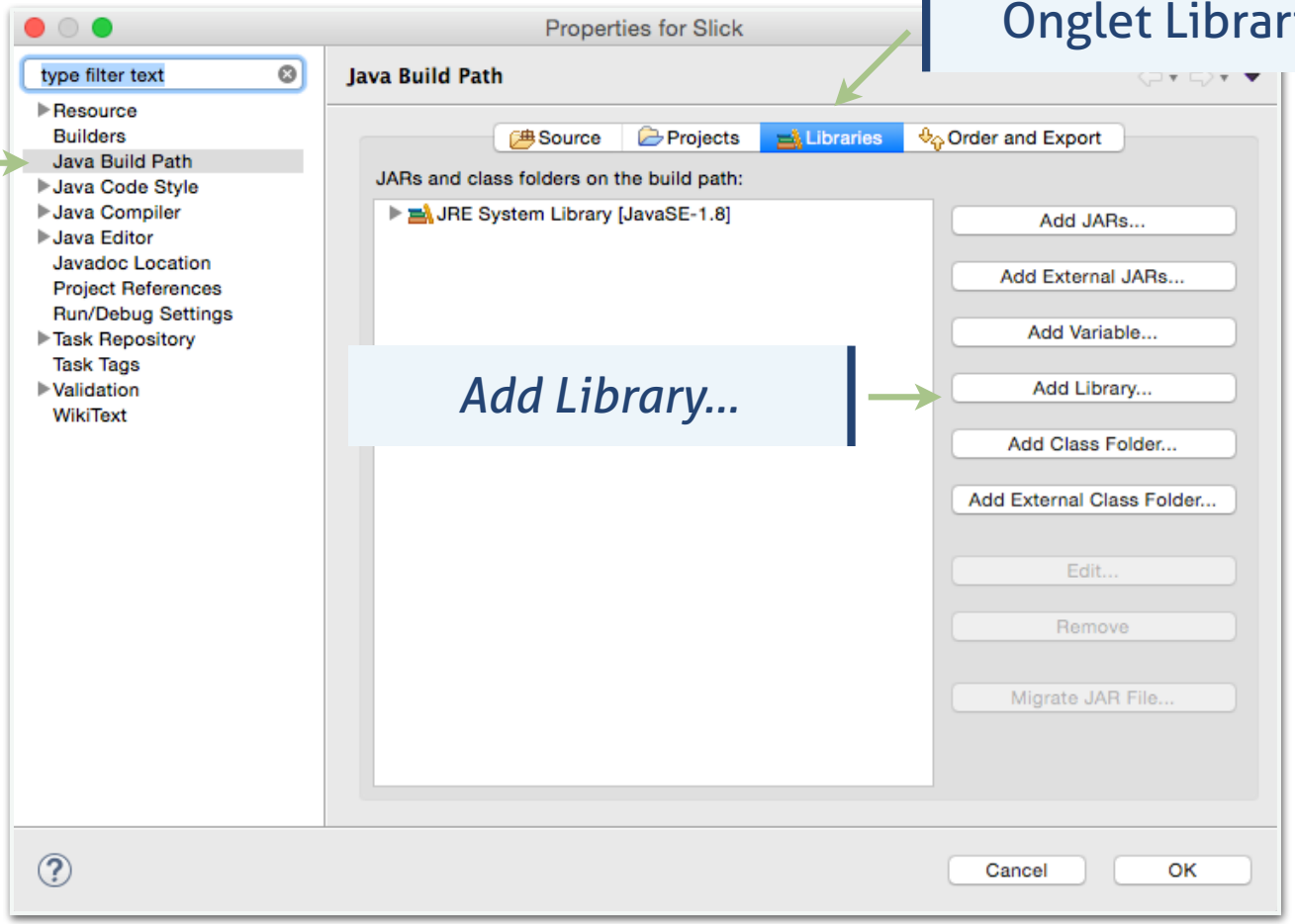
Doit indiquer le chemin de slick

CRÉATION D'UN PROJET AVEC SLICK 1/8

- Créer un projet java
- Intégrer bibliothèque Slick
  - Accéder au propriétés du projet : menu *Project* ➜ *Properties*
  - Dans la boîte de dialogue : *Java Build Path* puis l'onglet *Libraries*.
  - Cliquer sur *Add Library...*

Java Build Path

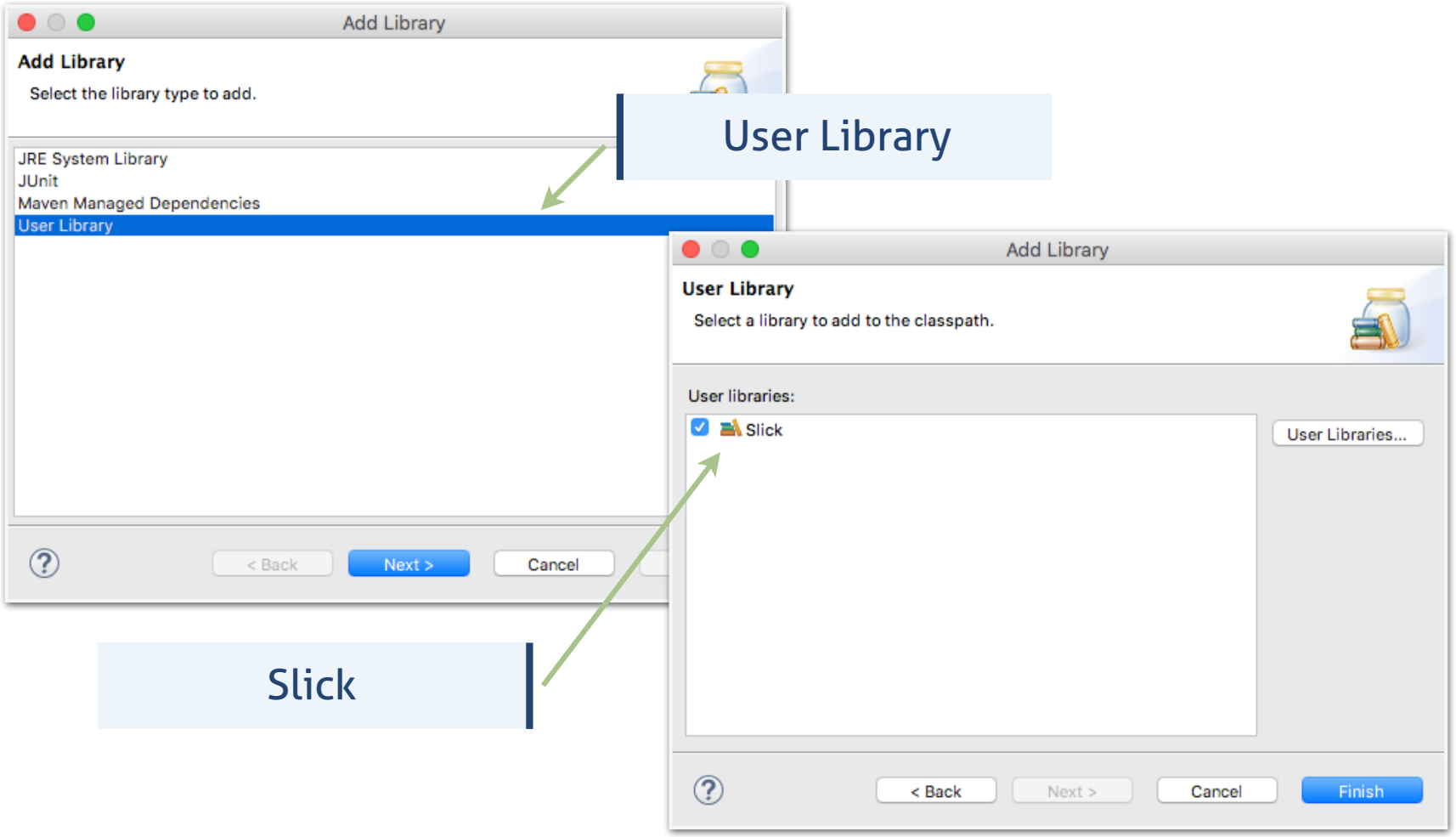
Onglet Libraries



Add Library...

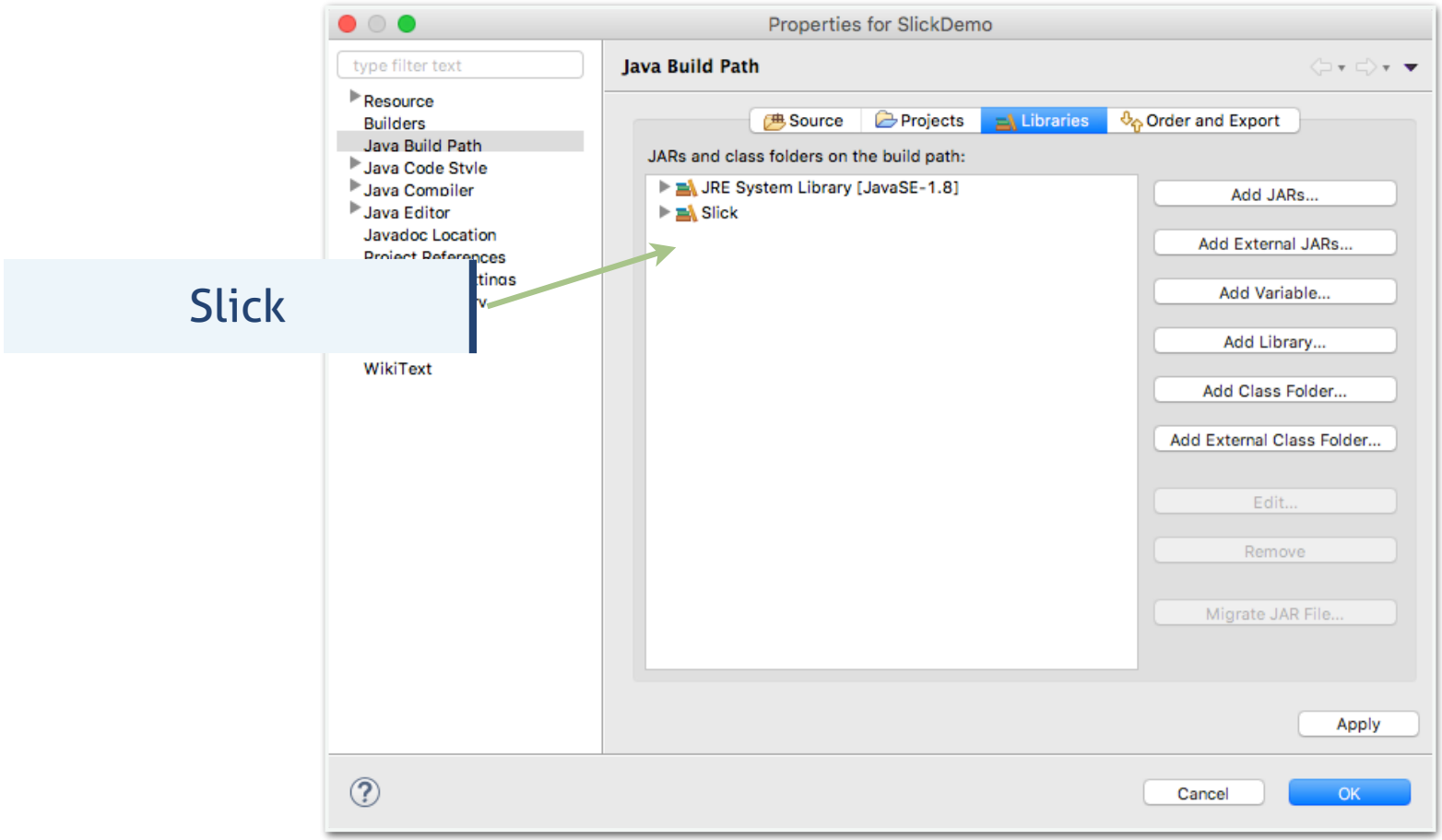
CRÉATION D'UN PROJET AVEC SLICK 2/8

- Sélectionner la bibliothèque Slick
  - Choisir *User Library*
  - Cocher *Slick*
  - Fermer



CRÉATION D'UN PROJET AVEC SLICK 3/8

- Fermer les propriétés du projet
  - La bibliothèque *Slick* doit faire partie de la liste

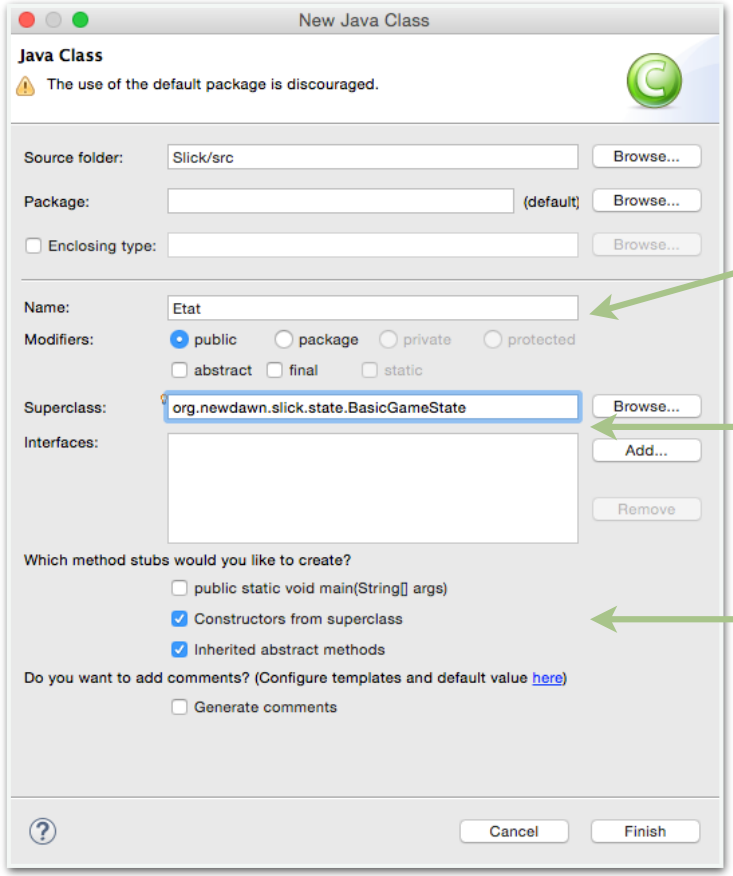


CRÉATION D'UN PROJET AVEC SLICK 4/8

- Jeu basé sur les états
  - Différentes façons de créer un jeu avec Slick.
  - Solution retenue ici : "jeu basé sur les états".
  - État : unité de découpage du jeu (écran de titre, niveau).
  - Minimum deux classes nécessaires :
    - Une classe dérivant de BasicGameState : un des états du jeu.
    - Une classe dérivant de StateBasedGame : classe de lancement du jeu.
- Important
  - Lors de la création des classes, cocher "Constructors from superclass" et "Inherited abstract methods"
  - Classes créées avec des méthodes prêtes à remplir.
  - Les classes mères viennent du package `org.newdawn.slick`.

CRÉATION D'UN PROJET AVEC SLICK 4/8

- Créer un état :
  - Initier la création d'une classe, nommée `Etat`
  - Dans la zone *Superclass*
    - Taper `BasicG`
    - Demander la complétion (CTRL-Espace), choisir `BasicGameState`
    - Le nom complet doit devenir `org.newdawn.slick.state.BasicGameState`.
  - Cocher les cases *Constructors form superclass* et *Inherited abstract methods*.



1 - Etat

2 - BasicGameState

3 - Cocher 2 cases

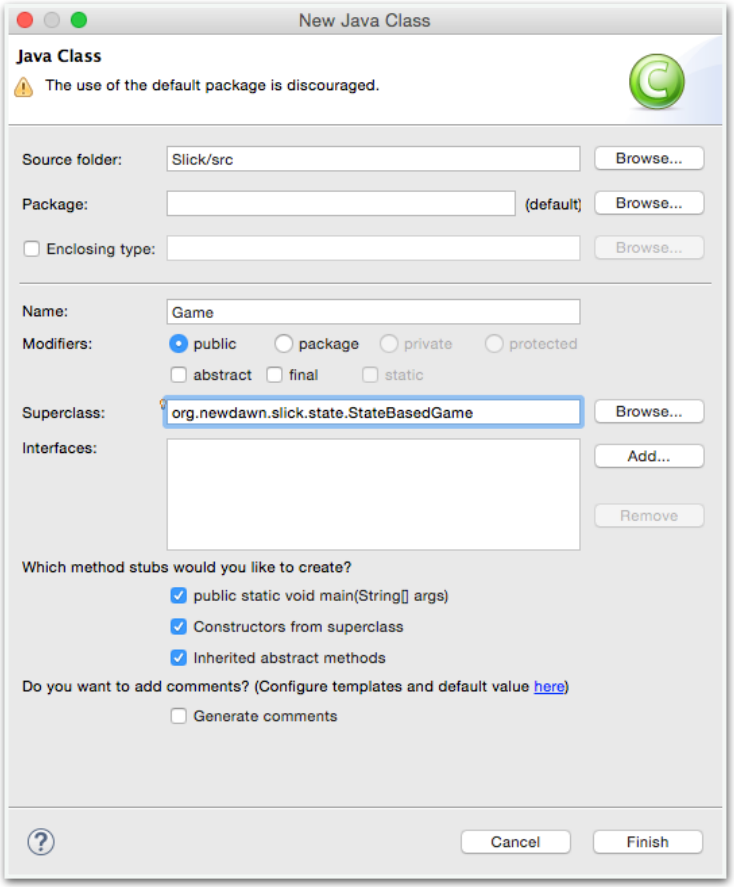
CRÉATION D'UN PROJET AVEC SLICK 5/8

- Créer la classe du jeu :
  - Initier la création d'une classe, nommée `Game`
  - Dans la zone *Superclass*
    - Taper `StateB`
    - Demander la complétion (CTRL-Espace), choisir `StateBasedGame`
    - Le nom complet doit devenir `org.newdawn.slick.state.StateBasedGame`.
  - Cocher les cases *Constructors form superclass* et *Inherited abstract methods*.

1 - Game

2 - StateBasedGame

3 - Tout chocer



CRÉATION D'UN PROJET AVEC SLICK 6/8

Éditer le fichier Game.java

- Modifier les méthodes `initStatesList()` et `main()`.

Game.java

```
01 @Override
02 public void initStatesList(GameContainer arg0) throws SlickException {
03     addState(new Etat());
04 }
05
06 public static void main(String[] args) {
07     AppGameContainer app;
08     try {
09         app = new AppGameContainer(new Game("jeu"));
10         app.setDisplayMode(800, 600, false);
11         app.setShowFPS(false);
12         app.start();
13     } catch(SlickException e){
14         e.printStackTrace();
15     }
16 }
```

Ajoute la classe Etat

Classe du jeu

Taille de la fenêtre

Démarre le jeu

- Lancer le programme, une fenêtre noire doit apparaître.

CRÉATION D'UN PROJET AVEC SLICK 7/8

Éditer la classe Etat

Renommer les paramètres des méthodes pour coller au rôle.

- Nommer `gc` les paramètres de classe `GameContainer`
- Nommer `sbg` les paramètres de classe `StateBasedGame`
- Nommer `g` le paramètre de classe `Graphics` (méthode `render()`).
- Nommer `delta` le paramètre de type `int` dans la méthode `update()`).

Etat.java

```
01 @Override
02 public void init(GameContainer gc, StateBasedGame sbg)
03         throws SlickException {
04 }
05
06 @Override
07 public void render(GameContainer gc, StateBasedGame sbg, Graphics g)
08         throws SlickException {
09 }
10
11 @Override
12 public void update(GameContainer gc, StateBasedGame sbg, int delta)
13         throws SlickException {
14 }
```

CRÉATION D'UN PROJET AVEC SLICK 8/8

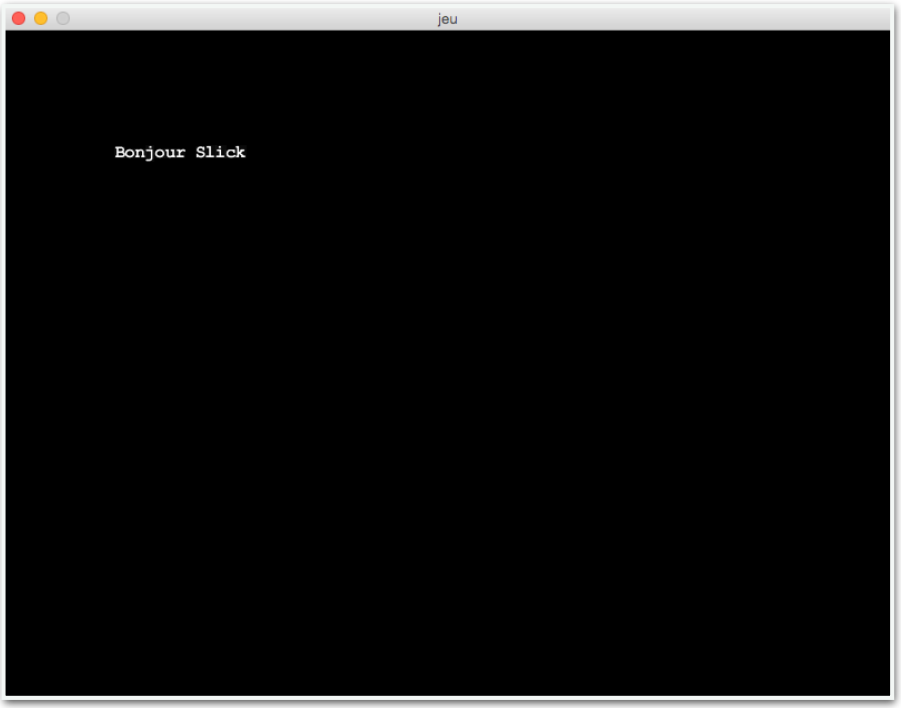
Éditer la méthode render()

Etat.java

```
01 public void render(GameContainer gc, StateBasedGame sbg, Graphics g)
02         throws SlickException {
03     g.drawString("Bonjour Slick", 100, 100);
04 }
```

Exécuter le programme

- La fenêtre noire doit présenter le message "Bonjour Slick".



EN CAS D'ERREURS

- Que faire ?
  - Vérifier les liaisons avec les fichiers .jar
  - Vérifier le chemin des bibliothèques natives.
  - Vérifier que les noms de classes correspondent bien aux classes créées.
  - Taper les noms des classes avec complétion, pour que les `import` s'ajoutent automatiquement.

IMPORTANT

UTILISER AU MAXIMUM LA COMPLÉTION, SURTOUT SUR LES NOMS DES CLASSES

- Beaucoup de classes à importer
  - Utilisation de très nombreuses classes
  - Permet d'éviter la complétion manuelle des classes

BOUCLE D'ETAT

Principe de fonctionnement

- Programme pour l'instant basé sur un seul état.
- Classe Game ouvre la fenêtre et démarre sur le premier état.

Boucle d'un état

- `init()` : réalise l'initialisation de l'état.
- `render()` : réalise l'affichage de l'état.
- `update()` : assure la mise à jour de l'état, possibilité d'y gérer les entrées (clavier, souris, joystick).

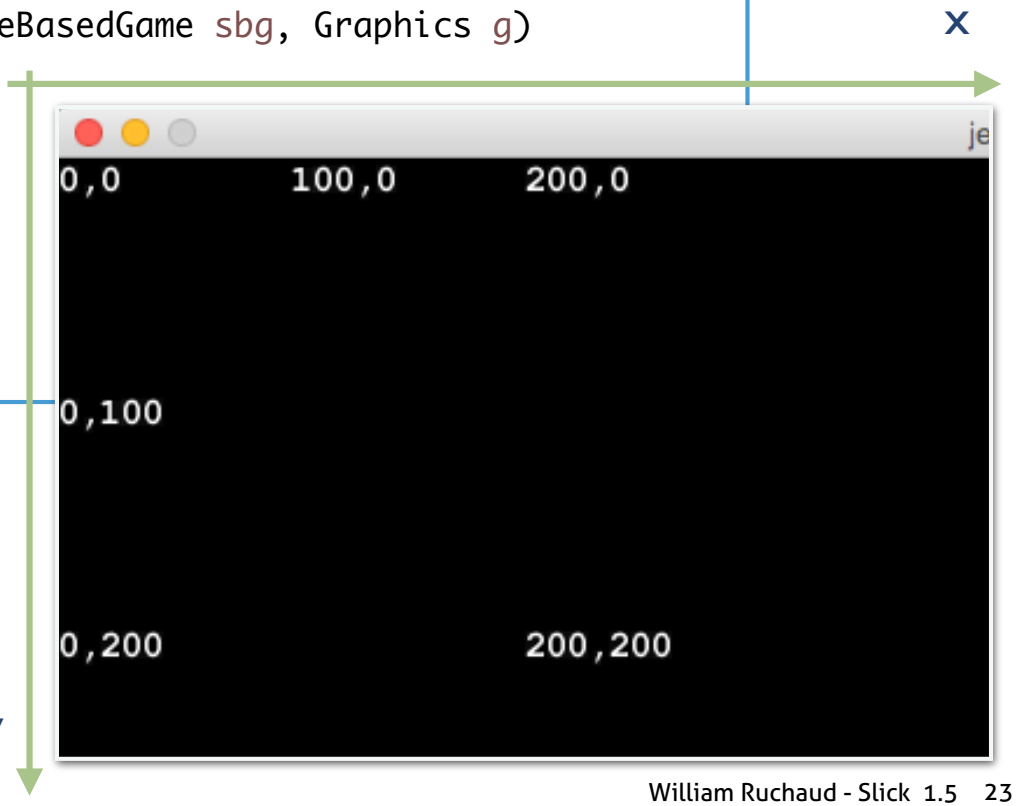
REPÈRE INFORMATIQUE

Inversion des y

- Affichage de l'écran par balayage de l'écran de gauche à droite et de haut en bas.
- Conséquences :
  - Localisation du repère (0,0) en haut à gauche de la fenêtre ou de l'écran.
  - Orientation des y positifs vers le bas, et des y négatifs vers le haut.
  - Point de référence des objets graphiques souvent en haut à gauche de la zone occupée.
- Coordonnées souvent gérer en float

Etat.java

```
01 @Override
02 public void render(GameContainer gc, StateBasedGame sbg, Graphics g)
03         throws SlickException {
04     g.drawString("0,0", 0, 0);
05     g.drawString("100,0", 100, 0);
06     g.drawString("200,0", 200, 0);
07     g.drawString("0,100", 0, 100);
08     g.drawString("0,200", 0, 200);
09     g.drawString("200,200", 200, 200);
10 }
```



LES PRIMITIVES DE DESSIN

Objet Graphics

- Objet Graphics doté d'un certain nombre de méthodes de dessin.
- Méthodes `drawXxx()` : dessin du contour d'une forme géométrique.
- Existence de plusieurs variantes (voir documentation).
- Épaisseur du tracé réglable avec la méthode `.setLineWidth()`.

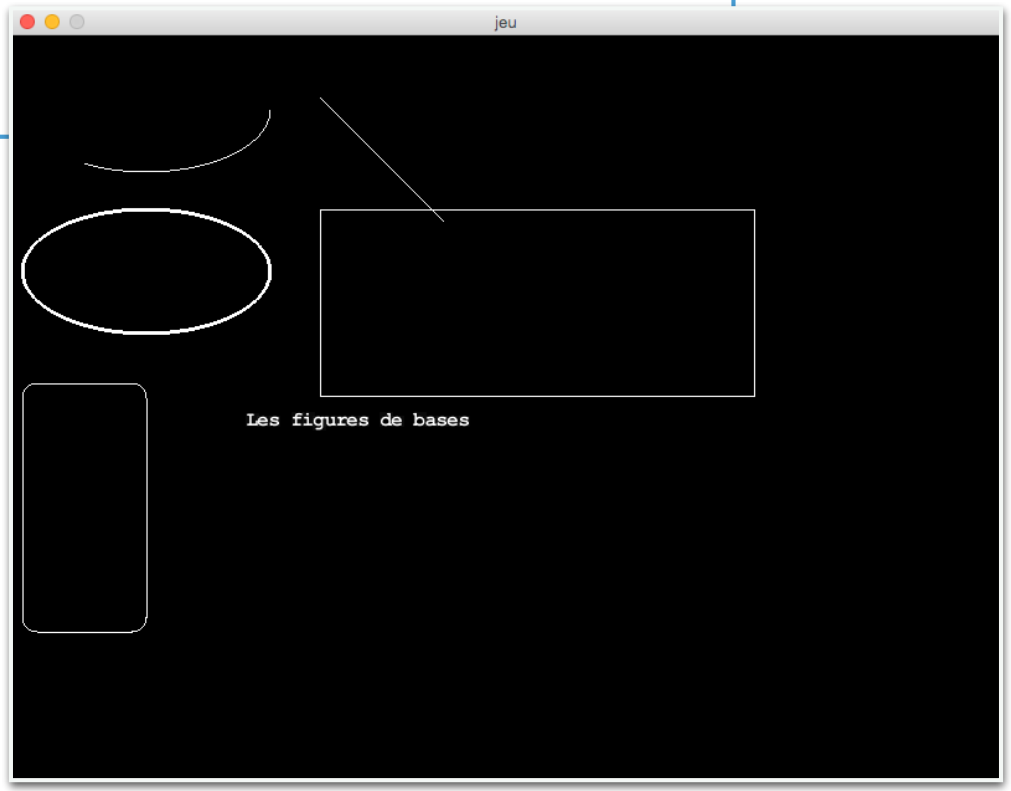
Méthode	Primitive de dessin
`drawArc(…)`	Arc de cercle
`drawLine(…)`	Segment de droite
`drawOval(…)`	Oval
`drawRect(…)`	Rectangle
`drawRoundRect(…)`	Rectangle arrondi
`drawString(…)`	Texte

LES PRIMITIVES DE DESSIN

Exemple

Etat.java

```
01 public void render(GameContainer gc, StateBasedGame sbg, Graphics g)
02         throws SlickException {
03         g.setLineWidth(1f);
04         g.drawString("Les figures de bases", 190, 300);
05         g.drawArc(10f,10f,200f,100f,0f,120f);
06         g.drawLine(250f, 50f, 350f, 150f);
07         g.drawRect(250f, 140, 350f, 150f);
08         g.drawRoundRect(10f, 280f, 100f, 200f, 12);
09         g.setLineWidth(3f);
10         g.drawOval(10f, 140f, 200f, 100f);
11 }
```

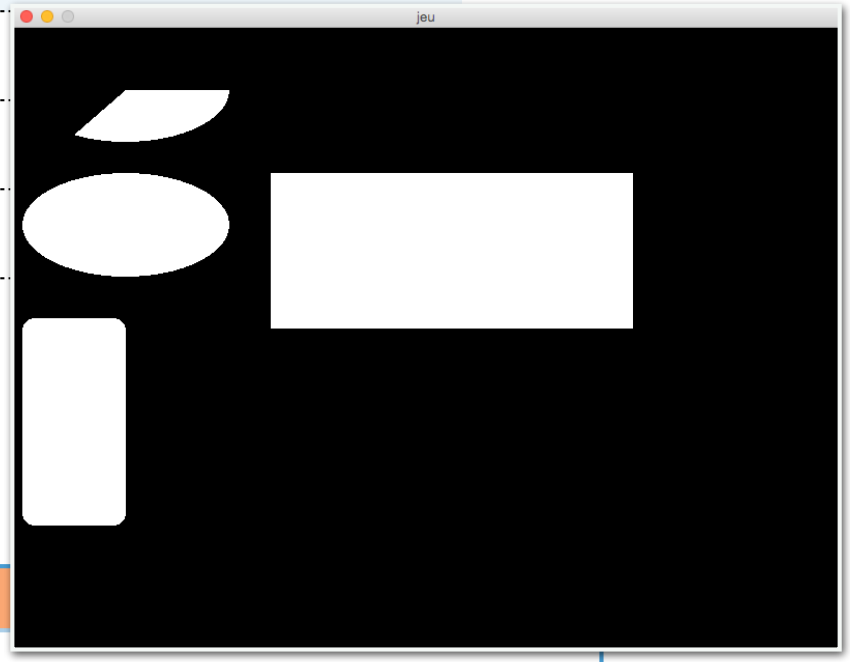


LES PRIMITIVES DE DESSIN

En mode plein

- Certaines primitives existent aussi en mode rempli

Méthode	Primitive de dessin
`fillArc(…)`	Arc de cercle
`fillOval(…)`	Oval
`fillRect(…)`	Rectangle
`fillRoundRect(…)`	Rectangle arrondi



Etat.java

```
01 @Override
02 public void render(GameContainer gc, StateBasedGame sbg, Graphics g)
03         throws SlickException {
04     g.fillArc(10f,10f,200f,100f,0f,120f);
05     g.fillRect(250f, 140, 350f, 150f);
06     g.fillRoundRect(10f, 280f, 100f, 200f, 12);
07     g.fillOval(10f, 140f, 200f, 100f);
08 }
```

GESTION DE LA COULEUR

- Classe Color
  - Gestion au niveau de l'objet `Graphics` avec `setColor()` et `getColor()`.
  - Couleurs codées en RGB (Rouge, Vert, Bleu) ou RGBA (Rouge, Vert, Bleu, Alpha) (Alpha = niveau de transparence).
  - 2 codages possibles :
    - Entiers : valeurs de 0 à 255.
    - Flottants : valeurs de 0f à 1f.
  - Exécuter `setColor()` avant l'opération de tracé.
  - Couleurs prédéfinies : `black`, `blue`, `cyan`, `yellow`, ...
- Couleur de fond
  - Propriété de l'objet `Graphics`.
  - Méthode `setBackground()` pour changer la couleur de fond.

Attention : importer org.newdawn.slick.Color

GESTION DE LA COULEUR

Exemple

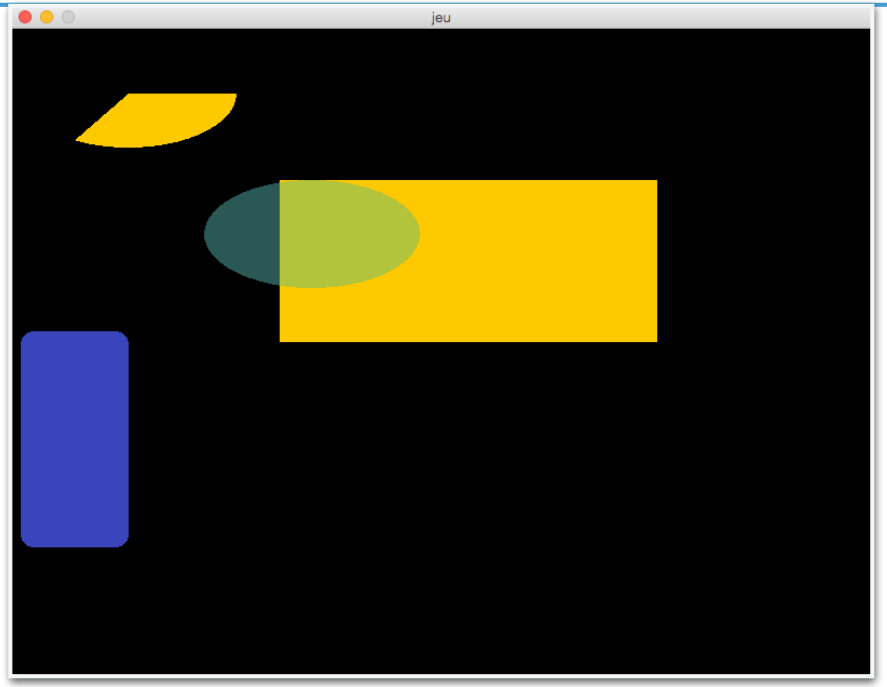
Etat.java

```
01 @Override
02 public void render(GameContainer gc, StateBasedGame sbg, Graphics g)
03         throws SlickException {
04     g.setColor(Color.orange);
05     g.fillArc(10f,10f,200f,100f,0f,120f);
06     g.fillRect(250f, 140, 350f, 150f);
07     g.setColor(new Color(60,70,180));
08     g.fillRoundRect(10f, 280f, 100f, 200f, 12);
09     g.setColor(new Color(90,190,180, 120));
10     g.fillOval(180f, 140f, 200f, 100f);
11 }
```

Couleur prédéfinie

Couleur personnalisée (int)

Couleur avec transparence



ORDRE D'EXÉCUTION

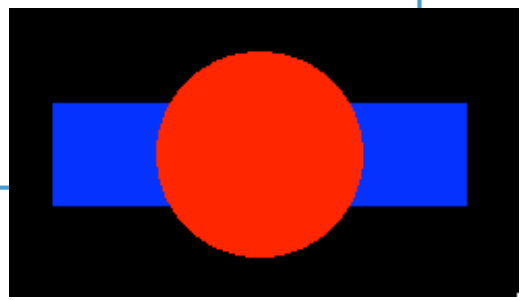
- Important : ordre d'exécution des primitives.
- Conséquence :
  - Possibilité de recouvrement.
  - Nécessité de procéder dans l'ordre en allant du fond vers le premier plan.

Etat.java (méthode render())

```
01  g.setColor(Color.red);
02  g.fillOval(100, 100, 100, 100);
03
04  g.setColor(Color.blue);
05  g.fillRect(50, 125, 200, 50);
```

Etat.java (méthode render())

```
01  g.setColor(Color.blue);
02  g.fillRect(50, 125, 200, 50);
03
04  g.setColor(Color.red);
05  g.fillOval(100, 100, 100, 100);
```



LES IMAGES

- Possibilité de charger des images :
  - Différents formats disponibles
  - Privilégier PNG (gestion de la transparence) ou JPEG (pas de transparence)
  - Regrouper les images dans un dossier (avec éventuellement des sous-dossiers)
- Principe :
  - Objet de classe Image pour contenir les données de l'image
  - Chargement de préférence dans la méthode `init()` (si image propre à l'état)
  - Attention au temps de chargement, surtout beaucoup d'images
  - Chargement pouvant échouer : lève une exception `SlickException`
  - Positionnement souvent par le coin supérieur gauche

Attention : importer org.newdawn.slick.Image

LES IMAGES

Chargement et affichages à partir de Graphics

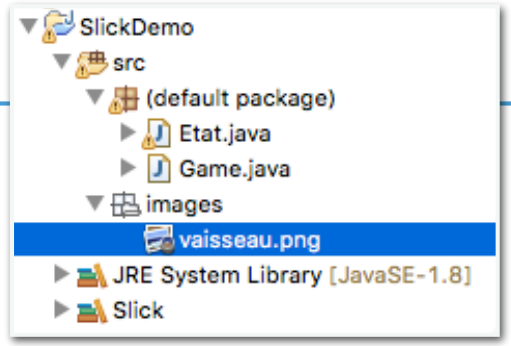
Etat.java

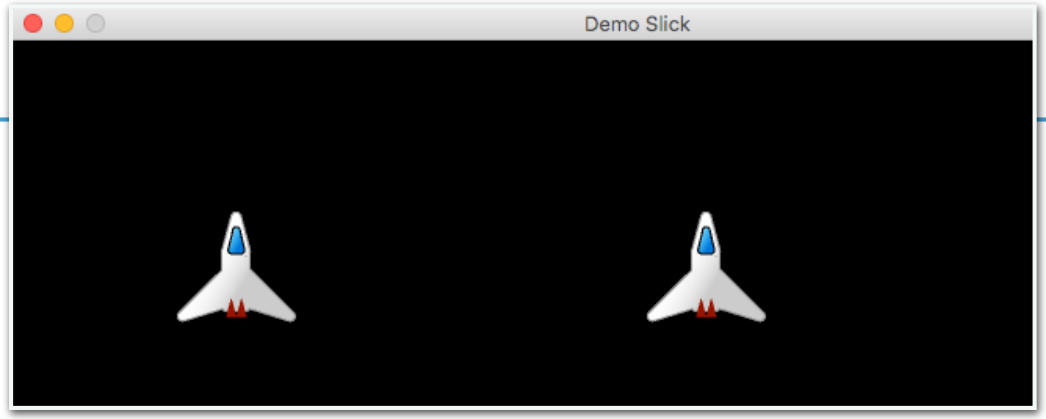
```
01 public class Etat extends BasicGameState {
02     protected Image spaceShip;
03
04 …
05
06     @Override
07     public void init(GameContainer gc, StateBasedGame sbg) throws SlickException {
08         spaceShip = new Image("images/vaisseau.png");
09     }
10
11     @Override
12     public void render(GameContainer gc, StateBasedGame sbg, Graphics g) throws
13 SlickException {
           g.drawImage(spaceShip, 100, 100);
14         g.drawImage(spaceShip, 400, 100);
15     }
16
   …
   }
```

Propriété

Chargement

Affichages





LES IMAGES : AFFICHAGE

Chargement et affichages à partir de Image

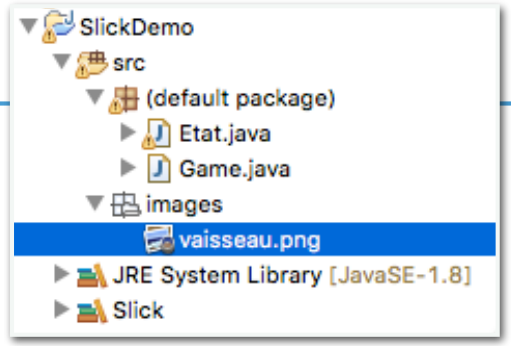
Etat.java

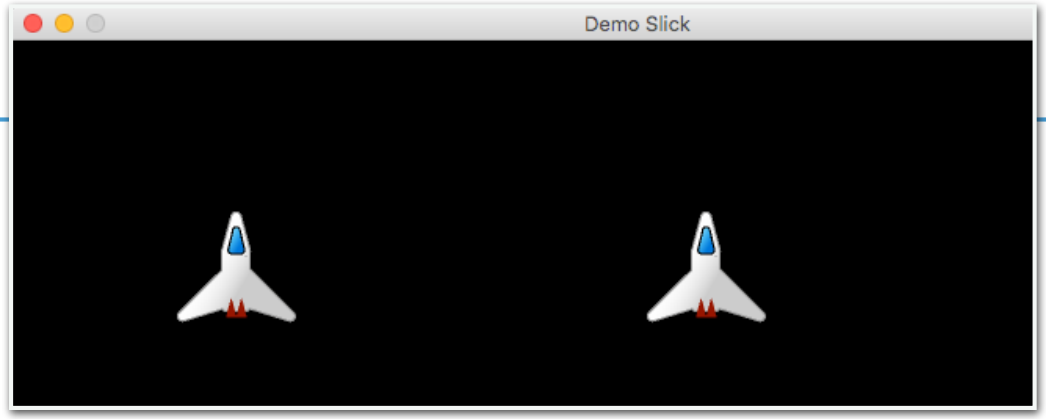
```
01 public class Etat extends BasicGameState {
02     protected Image spaceShip;
03
04 …
05
06     @Override
07     public void init(GameContainer gc, StateBasedGame sbg) throws SlickException {
08         spaceShip = new Image("images/vaisseau.png");
09     }
10
11     @Override
12     public void render(GameContainer gc, StateBasedGame sbg, Graphics g) throws
13 SlickException {
           spaceShip.draw(100, 100);
14         spaceShip.draw(400, 100);
15     }
16
   …
   }
```

Propriété

Chargement

Affichages





IMAGES : AFFICHAGE

Quelques variantes sur les méthodes de Images

Effet de couleur

```
01 spaceShip.draw(100, 100, Color.red);
02 spaceShip.draw(400, 100);
```

Mise à l'échelle uniforme

```
01 spaceShip.draw(100, 100, 0.5f);
02 spaceShip.draw(400, 100);
```

Mise à l'échelle différenciée

```
01 spaceShip.draw(100, 100, 176, 85);
02 spaceShip.draw(400, 100);
```

Affichage partiel (possibilité de mise à l'échelle)

```
01 spaceShip.draw(100, 100, 130, 160, 20, 0, 50, 60);
02 spaceShip.draw(400, 100);
```

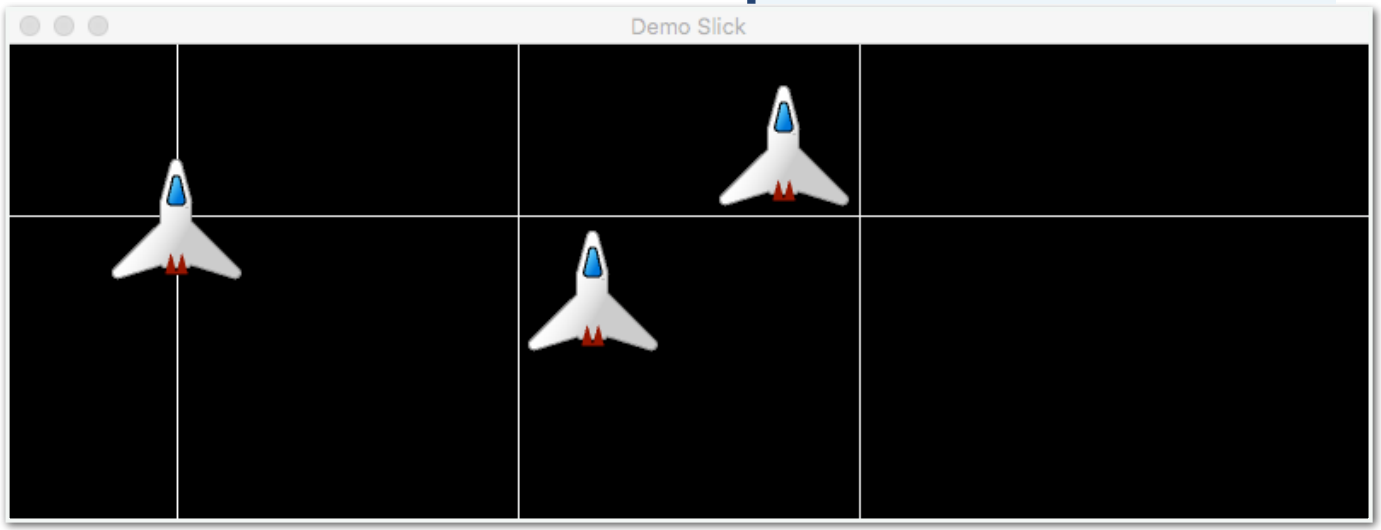
IMAGES : AFFICHAGE

Méthode drawCentered()

- Attention : ne se base pas sur le contenu de l'image, mais sur le rectangle englobant.

```
01 g.drawLine(100, 0, 100, 800);
02 g.drawLine(300, 0, 400, 800);
03 g.drawLine(500, 0, 500, 800);
04 g.drawLine(0, 100, 800, 100);
05 spaceShip.drawCentered(100, 100);
06 spaceShip.draw(300, 100);
07 spaceShip.draw(500-spaceShip.getWidth(), 100-spaceShip.getHeight());
```

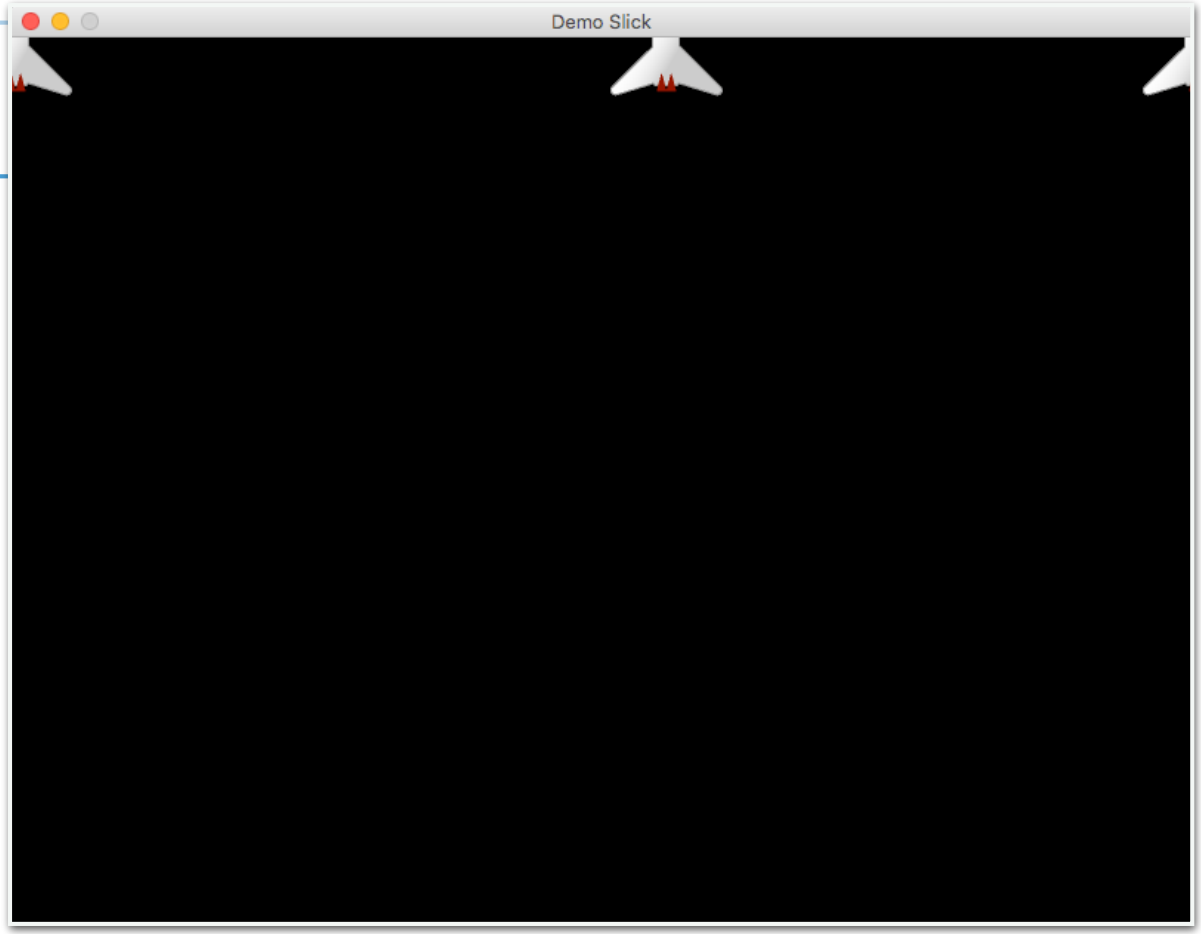
Positionnement à partir du coin bas-droite



IMAGES : AFFICHAGE

- Affichage hors-zone
  - Coordonnées virtuellement infinies
  - Affichage possible en dehors de la zone visible ou à cheval

```
01 spaceShip.draw(-40, -40);
02 spaceShip.draw(400, -40);
03 spaceShip.draw(760, -40);
```

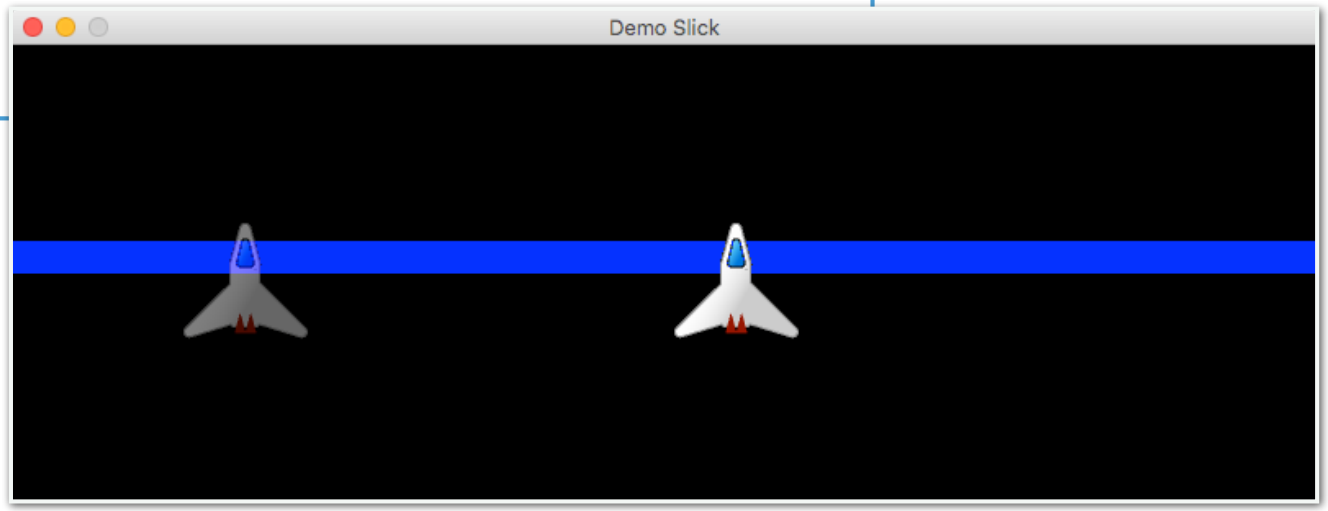


IMAGES : TRANSPARENCE

Méthode setAlpha()

- Permet de régler la transparence globale de l'image.
- Valeur entre 0.0f (complètement transparent) à 1.0f (opaque)

```
01 g.setColor(Color.blue);
02 g.fillRect(0, 120, 800, 20);
03 spaceShip.setAlpha(0.5f);
04 spaceShip.draw(100, 100);
05 spaceShip.setAlpha(1.0f);
06 spaceShip.draw(400, 100);
```

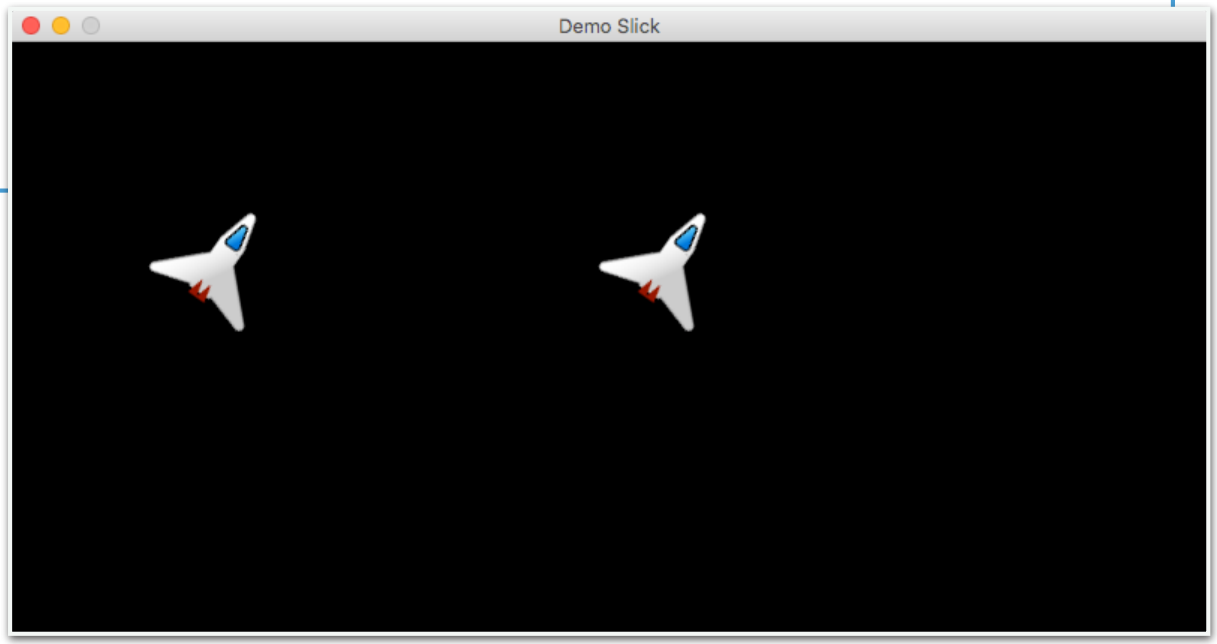


IMAGES : ROTATION

- Méthode rotate() :
  - Rotation de l'objet image
  - Attention : s'applique à tous les affichages (possibilité de créer des copies)

Etat.java

```
01 @Override
02 public void init(GameContainer gc, StateBasedGame sbg) throws SlickException {
03     spaceShip = new Image("images/vaisseau.png");
04     spaceShip.rotate(35f);
05 }
06
07 @Override
08 public void render(GameContainer gc, StateBasedGame sbg, Graphics g) throws
09 SlickException {
10     spaceShip.draw(100, 100);
11     spaceShip.draw(400, 100);
12 }
```



ANIMATION

- Utilisation combinée de update() et render()
  - `render()` ne doit s'occuper que de l'affichage
  - Modification des coordonnées à effectuer dans `update()`
  - Paramètre `delta` de `update()` = temps écoulé en millisecondes depuis le dernier appel de `update()`
- Attention : framerate variable !!!
  - Framerate : nombre d'images par seconde
  - Solution : avoir une vitesse en pixels par seconde et calculer le déplacement en fonction du `delta` de `update()`

ANIMATION

Exemple d'animation simple

Etat.java

```
01 public class Etat extends BasicGameState {
02     protected Image spaceShip;
03     protected float y = 500;
04     protected float vY = -200;
05 …
06
07     @Override
08     public void render(GameContainer gc, StateBasedGame sbg, Graphics g) throws
   SlickException {
09         spaceShip.draw(100, y);
10     }
11
12     @Override
13     public void update(GameContainer gc, StateBasedGame sbg, int delta) throws
   SlickException {
14         y = y + delta * vY / 1000f;
15         if(y<-spaceShip.getHeight()) {
16             y = gc.getHeight();
17         }
18     }
       …
   }
```

y = position vY = vitesse

Fait avancer le vaisseau et le replace en bas de l'écran s'il "sort" en haut

GESTION DES ENTRÉES

- Entrée : souris, clavier, joystick
- Deux types de détections possibles :
  - Détection avec effet ponctuel (événement) :
    - Action déclenchée au moment de l'appui (ou du relâchement)
    - Pas de poursuite de l'effet quand la touche reste enfoncée
  - Détection avec effet continu (polling) :
    - Vérification de l'état des touches, boutons, etc...

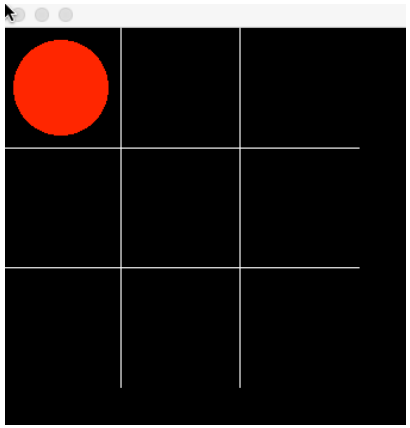
- Deux possibilités de gestion des entrées
  - Par des méthodes dédiées : détection avec effet ponctuel uniquement
  - Par la classe `Input` dans la méthode `update()` : pour les deux types

GESTION DES ENTRÉES

Par des méthodes spécifiques

Etat.java

```
01 public class Etat extends BasicGameState {
02     protected int x = 0;
03     protected int y = 0;
04
05 …
06     @Override
07     public void render(GameContainer gc, StateBasedGame sbg, Graphics g) throws
08 SlickException {
           g.setColor(Color.white);
09         g.drawLine(100, 0, 100, 300);
10         g.drawLine(200, 0, 200, 300);
11         g.drawLine(0, 100, 300, 100);
12         g.drawLine(0, 200, 300, 200);
13         g.setColor(Color.red);
           g.fillOval(x*100+10,y*100+10,80,80);
14     }
15 …
16     @Override
17     public void keyPressed(int key, char c) {
18         super.keyPressed(key, c);
           if(key==Keyboard.KEY_LEFT && x>0) x--;
           if(key==Keyboard.KEY_RIGHT && x<2) x++;
           if(key==Keyboard.KEY_UP && y>0) y--;
           if(key==Keyboard.KEY_DOWN && y<2) y++;
       }
   }
```



Méthode déclenchée à chaque appui sur une touche

Classe contenant les constantes des touches

GESTION DES ENTRÉES

- Liste des principales méthodes

Méthode	Événement
`keyPressed(…)`	Touche enfoncée
`keyReleased(…)`	Touche relâchée
`mouseClicked(…)`	Clic de la souris (gère double clic)
`mouseDragged(…)`	Glissé déposé (bouton enfoncé)
`mouseMoved(…)`	Mouvement de la souris
`mousePressed(…)`	Bouton de la souris enfoncé
`mouseReleased(…)`	Bouton de la souris relâché
`mouseWheelMoved(…)`	Mouvement de la molette

- Existence de méthodes dédiées aux joysticks

GESTION DES ENTRÉES

- Dans la méthode update()
  - Utilisation de l'objet Input.
  - S'obtient à partir de l'objet GameContainer
  - Dispose d'un grand nombre de méthodes

Etat.java

```
01 public void update(GameContainer gc, StateBasedGame sbg, int delta) throws SlickException
02 {
03     Input input = gc.getInput();
04     if(input.isKeyPressed(Input.KEY_LEFT) && x>0) x--;
05     if(input.isKeyPressed(Input.KEY_RIGHT) && x<2) x++;
06     if(input.isKeyPressed(Input.KEY_UP) && y>0) y--;
07     if(input.isKeyPressed(Input.KEY_DOWN) && y<2) y++;
08 }
```

GESTION DES ENTRÉES

Quelques méthodes de Input

Méthode	Événement
`getMouseX()`	Abscisse de la souris
`getMouseY()`	Ordonnée de la souris
`isKeyDown(…)`	Touche enfoncée (continu)
`isKeyPressed(…)`	Touche enfoncée (événement)
`isMouseButtonDown(…)`	Bouton de la souris enfoncé (continu)
`isMousePressed(…)`	Bouton de la souris enfoncé (événement)

GESTION DES ENTRÉES

Constantes de Input ou de Keyboard

Description	Constantes
Chiffres	KEY_0, KEY_1, … KEY_9
Lettres	KEY_A, KEY_B, … KEY_Z
Touches du curseur	KEY_LEFT, KEY_UP, KEY_RIGHT,
Barre espace	KEY_SPACE
Touche échappe	KEY_ESCAPE
Touche entrée	KEY_RETURN
Touches du pavé	KEY_NUMPAD0, KEY_NUMPAD1,
Touches ALT	KEY_LALT, KEY_RALT
Touches contrôle	KEY_LCONTROL,
Boutons de la souris	MOUSE_LEFT_BUTTON, MOUSE_RIGHT_BUTTON,

UTILISATION DES POLICES

- drawString() limité
  - Police et taille prédéfinie

- Solutions :
  - Usage du texte limité : utiliser des images
  - Usage de beaucoup de texte : charger une police (TrueType ou Unicode)

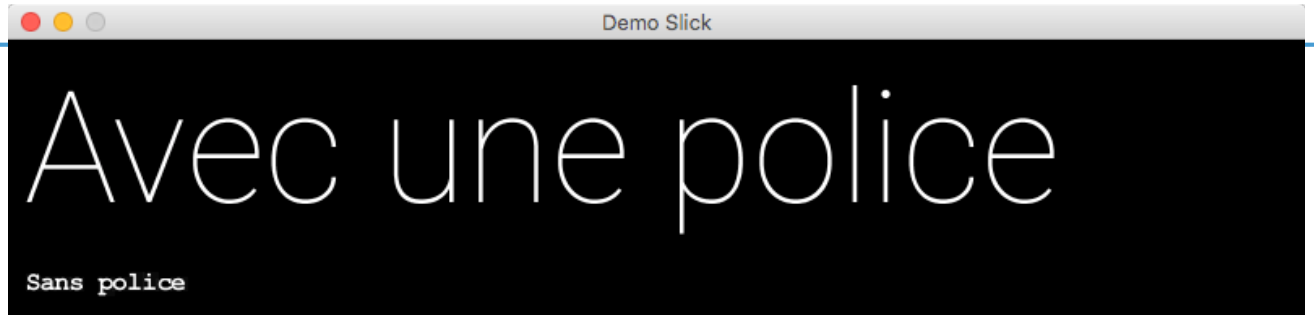
- Limite des polices
  - Mise en cache des caractères pour une taille donnée
  - Se limiter à quelques tailles précises

UTILISATION DES POLICES

Exemple

Etat.java

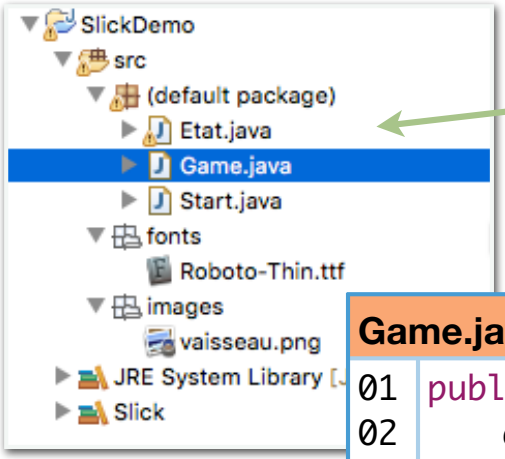
```
01 public class Etat extends BasicGameState {
02     protected UnicodeFont roboto;
03 
04     public Etat() {
05     }
06 
07     @SuppressWarnings("unchecked")
08     @Override
09     public void init(GameContainer gc, StateBasedGame sbg) throws SlickException {
10         roboto = new UnicodeFont("fonts/Roboto-Thin.ttf",96,false,false);
11         roboto.addAsciiGlyphs();
12         roboto.getEffects().add(new ColorEffect());
13         roboto.loadGlyphs();
14     }
15 
16     @Override
17     public void render(GameContainer gc, StateBasedGame sbg, Graphics g) throws
   SlickException {
18         roboto.drawString(10, 10, "Avec une police", Color.white);
19         g.drawString("Sans police", 10, 140);
20     }
```



GESTION DES ÉTATS

- État = niveau ou écran
  - Rendu graphique + gestion différente
  - Permet de découper le jeu en morceaux
  - Exemple : écran de présentation, jeu, score

- 1 état = 1 classe enfant de BasicGameState
  - Bien renseigner la méthode `getID()` avec un numéro différent pour chaque état
  - Penser à ajouter tous les états dans `initStatesList()` de la classe de démarrage



Etat.java et Start.java sont des états

Game.java

```
01 public void initStatesList(GameContainer arg0) throws SlickException {
02     addState(new Etat());
03     addState(new Start());
04 }
```

GESTION DES ÉTATS

- Changer d'état
  - Appeler la méthode `enterState()` de la classe `StateBasedGame`
  - Se fait généralement dans `update()`