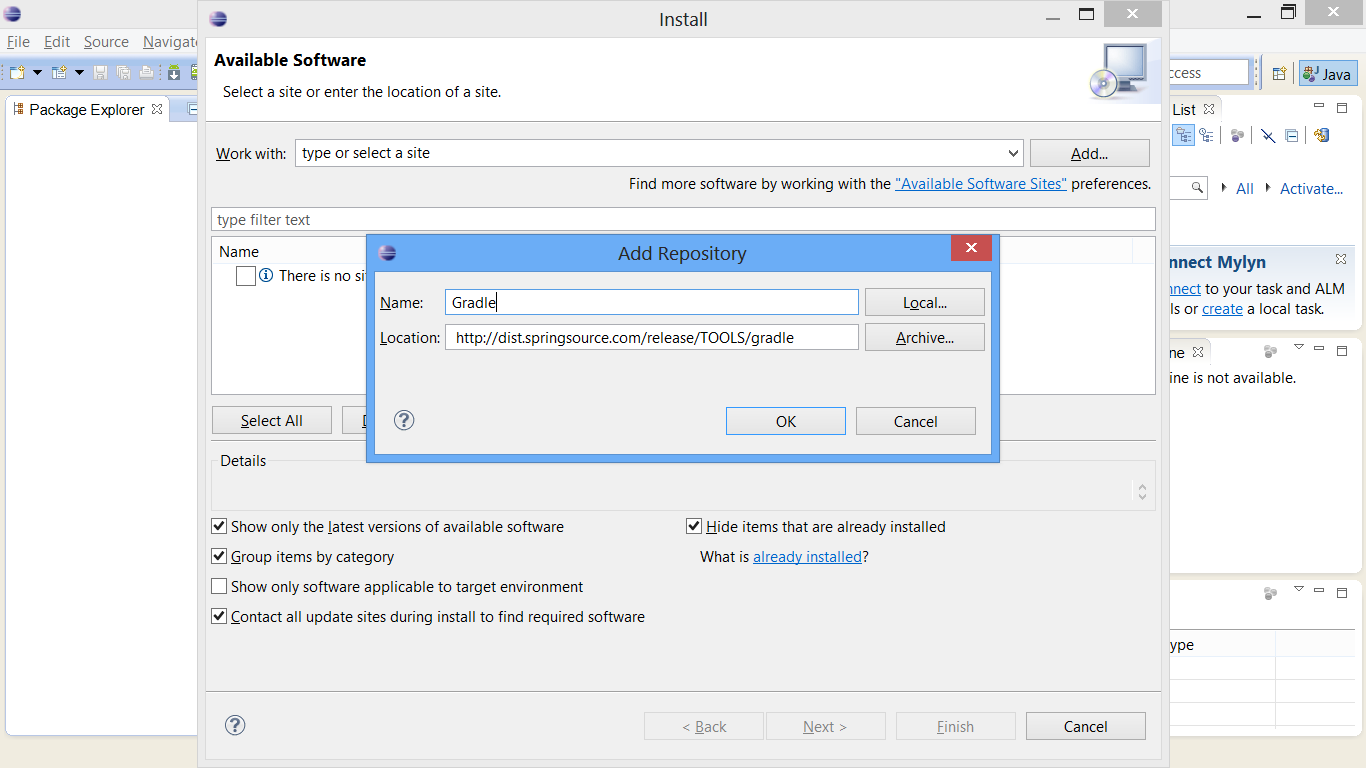
Viết game con chim điên trên libGDX framework

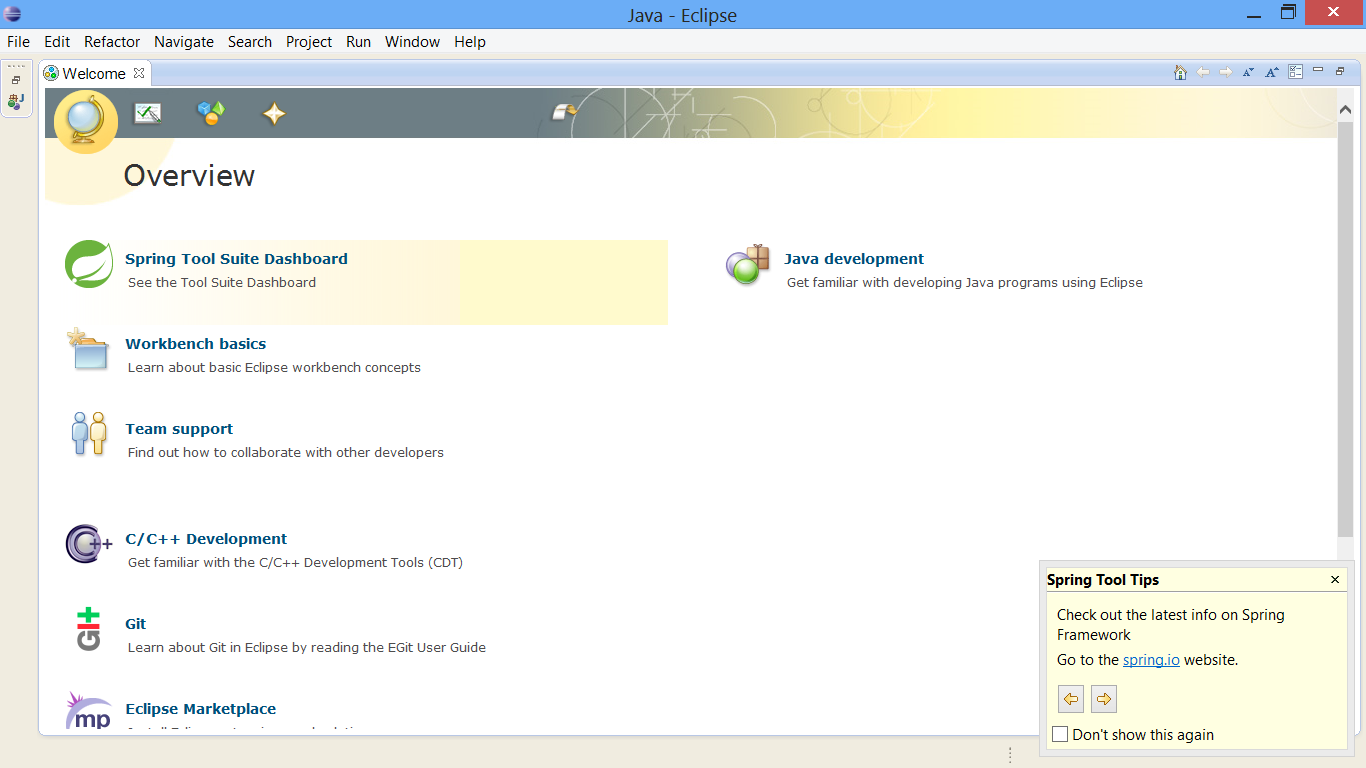
Game con chim sẽ được viết bằng libGDX framework. Framework này khá mạnh, viết trên java nhưng có thể dịch ra android, ios, desktop, html5, dâu đen. Tham khảo framework tại địa chỉ <http://libgdx.badlogicgames.com/>

# Cài Gradle thêm vào Eclipse

1. Vào Eclipse chọn Help->Install new software, nhấn nút Add nhập vào mục name là Gradle và dán địa chỉ <http://dist.springsource.com/release/TOOLS/gradle> vào phần Location rồi nhấn OK.

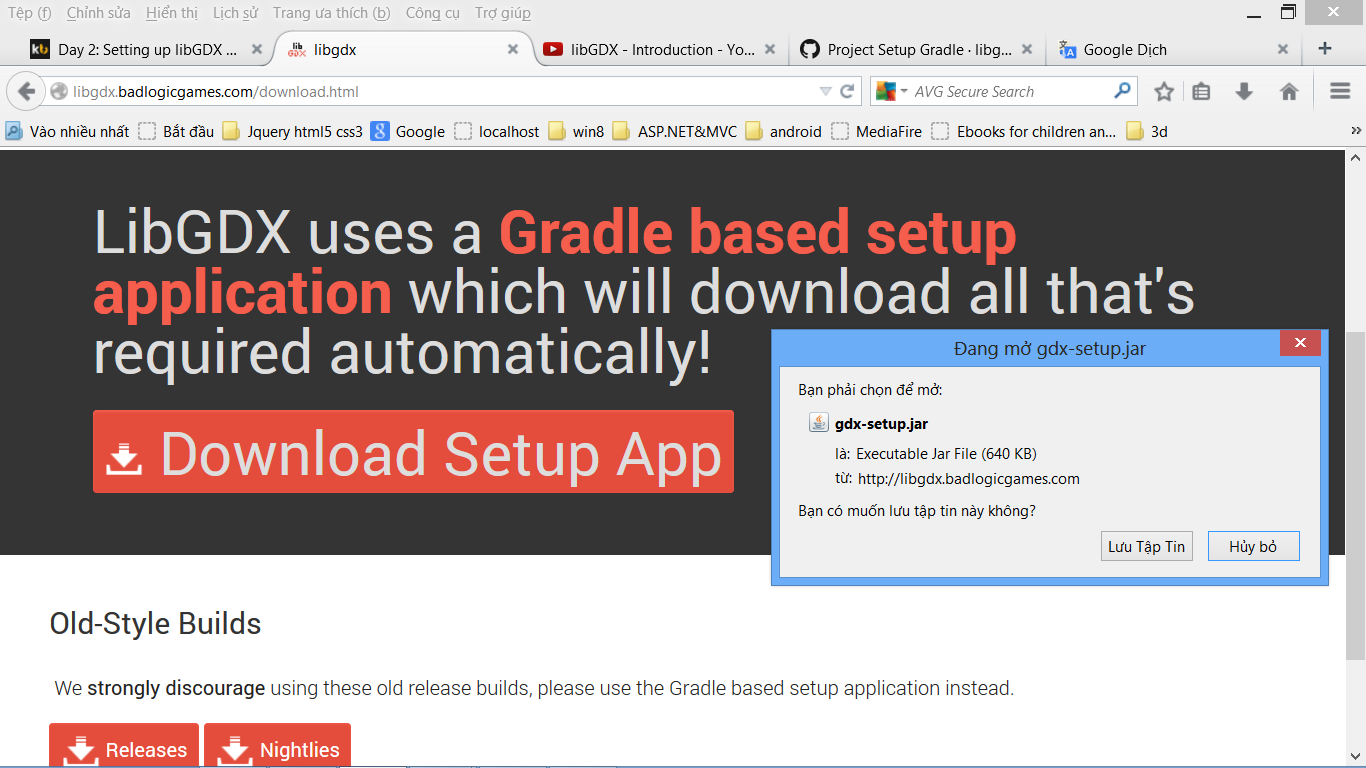


1. Sau khi nó quét xong nhấn select all rồi nhấn next -> next chọn checkbox chấp nhận rồi nhấn Finish. Chương trình sẽ cài đặt. Sau khi cài xong nó sẽ kêu khởi động lại eclipse.
2. Khi mở eclipse lên sẽ có thêm Spring Tool Suite Dashboard.

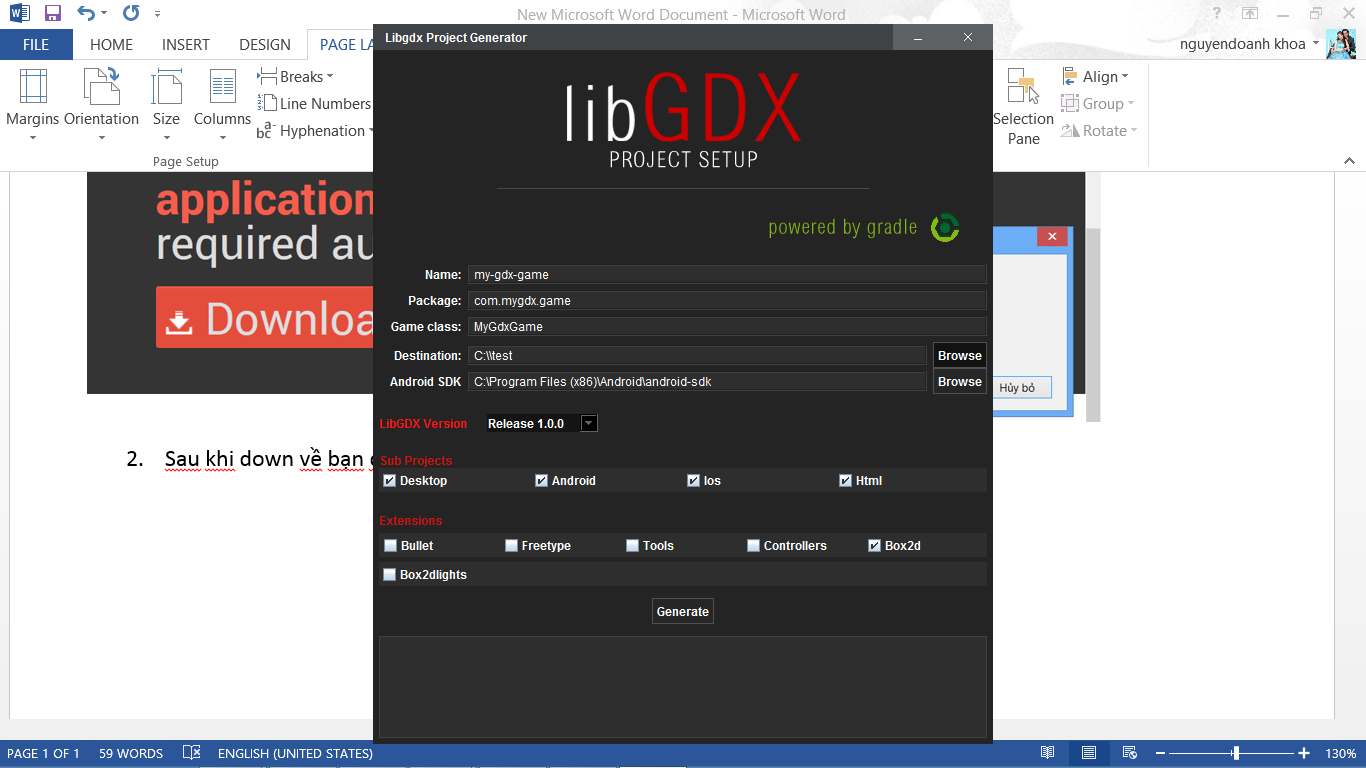


# Tạo Project theo libgdx

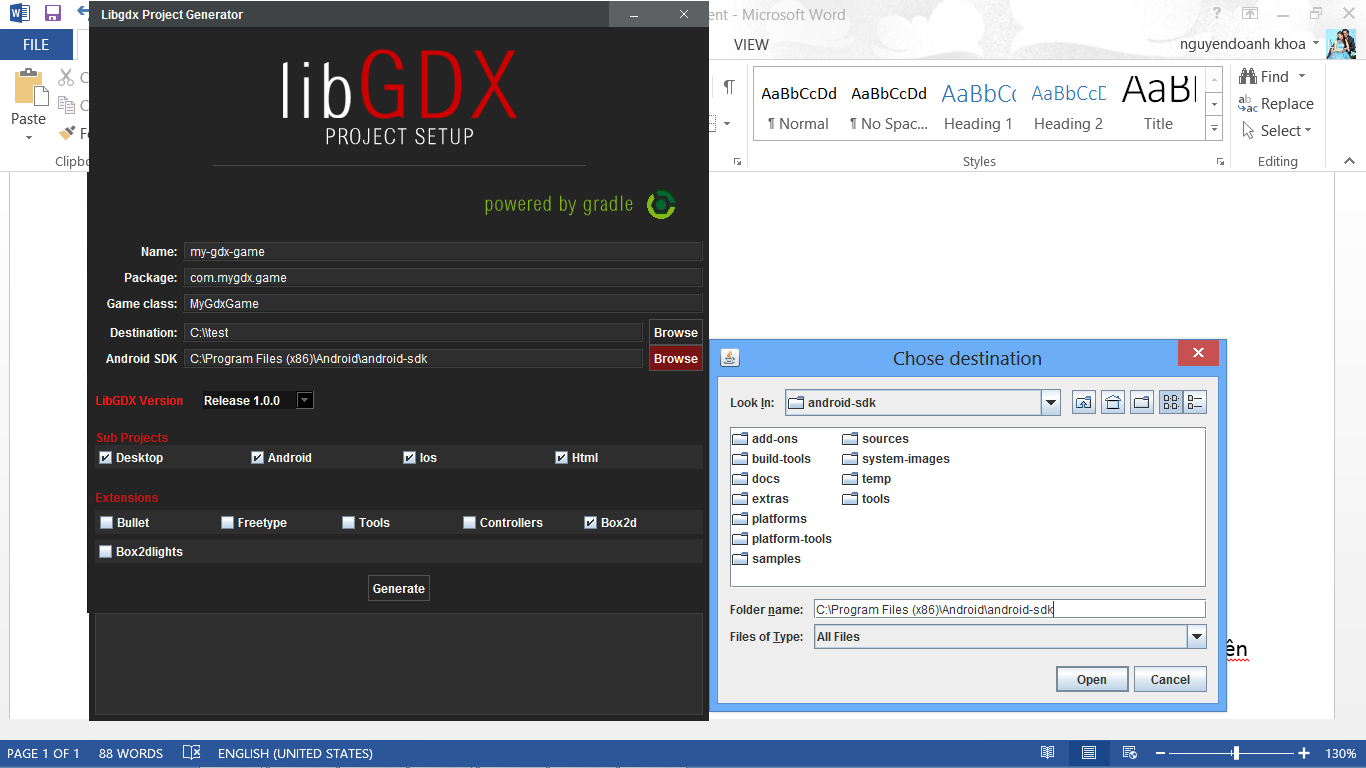
1. Truy cập địa chỉ <http://libgdx.badlogicgames.com/download.html> chọn nút download.



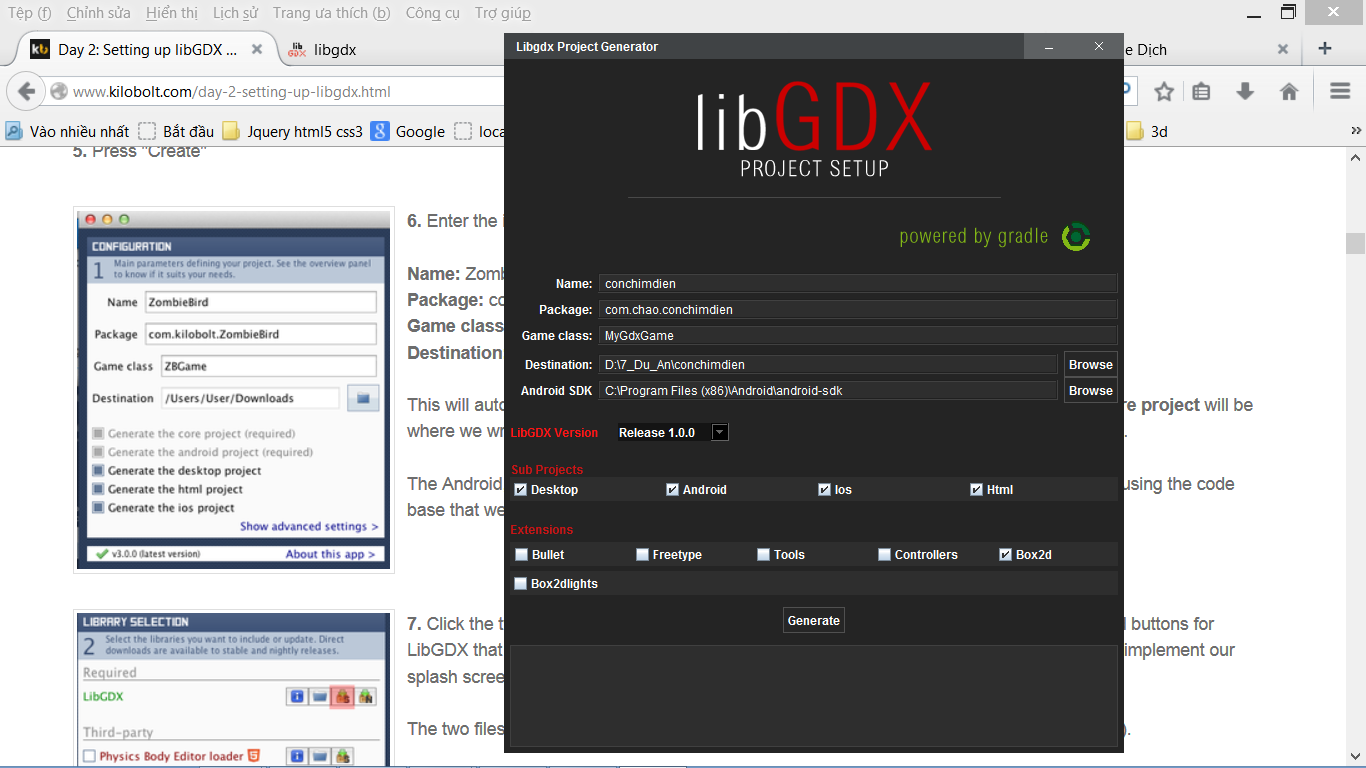
1. Sau khi down về bạn được 1 file gdx-setup.jar, double click được hình như sau,



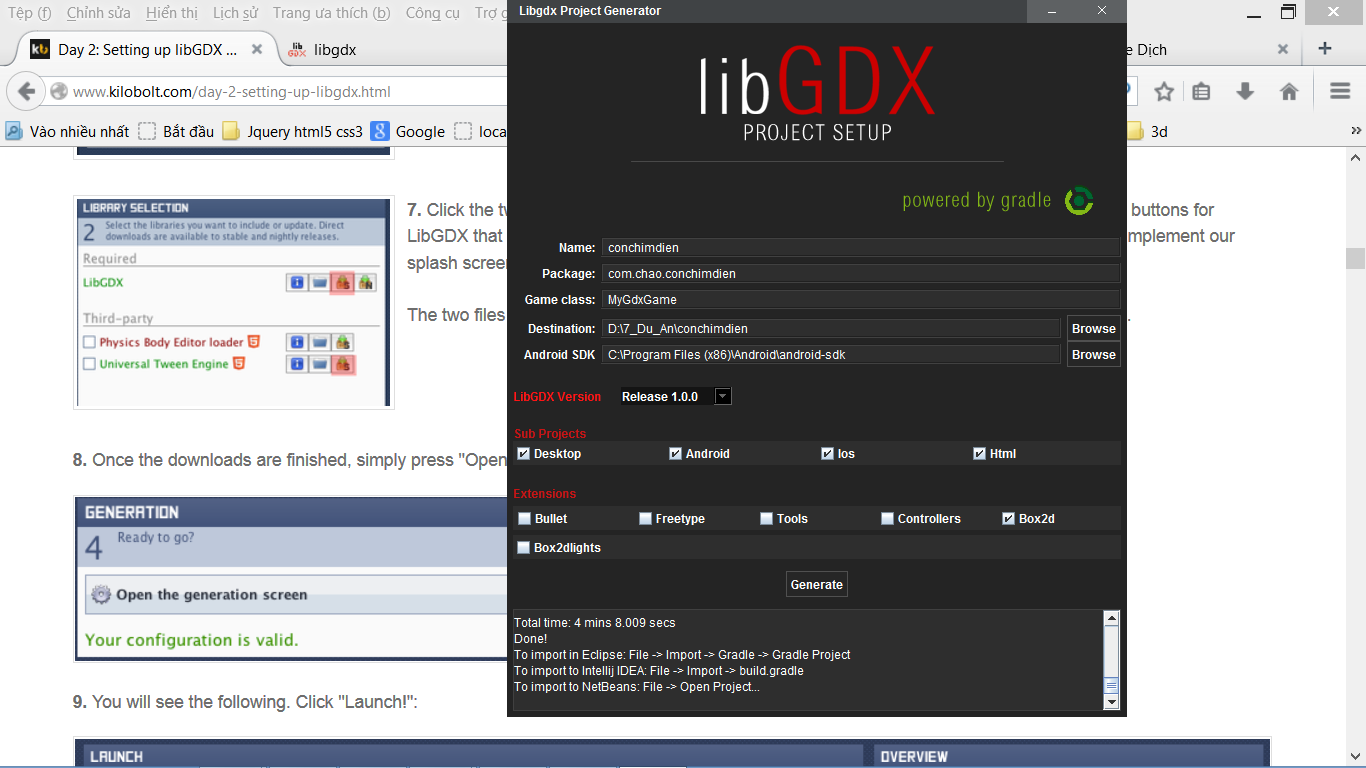
1. Chú ý ở bước này bạn phải chọn vào nút browse ở mục Android SDK và chỉ vào thư mục Android SDK đã cài trên máy của bạn như hình dưới.



1. Sau đó bạn còn phải nhập thêm các thông tin về tên project, tên package, destination(vị trí lưu project) vào vd như hình bên dưới sau đó nhấn nút “Generate” bên dưới rồi ngồi chờ nó download về cũng khá là lâu:

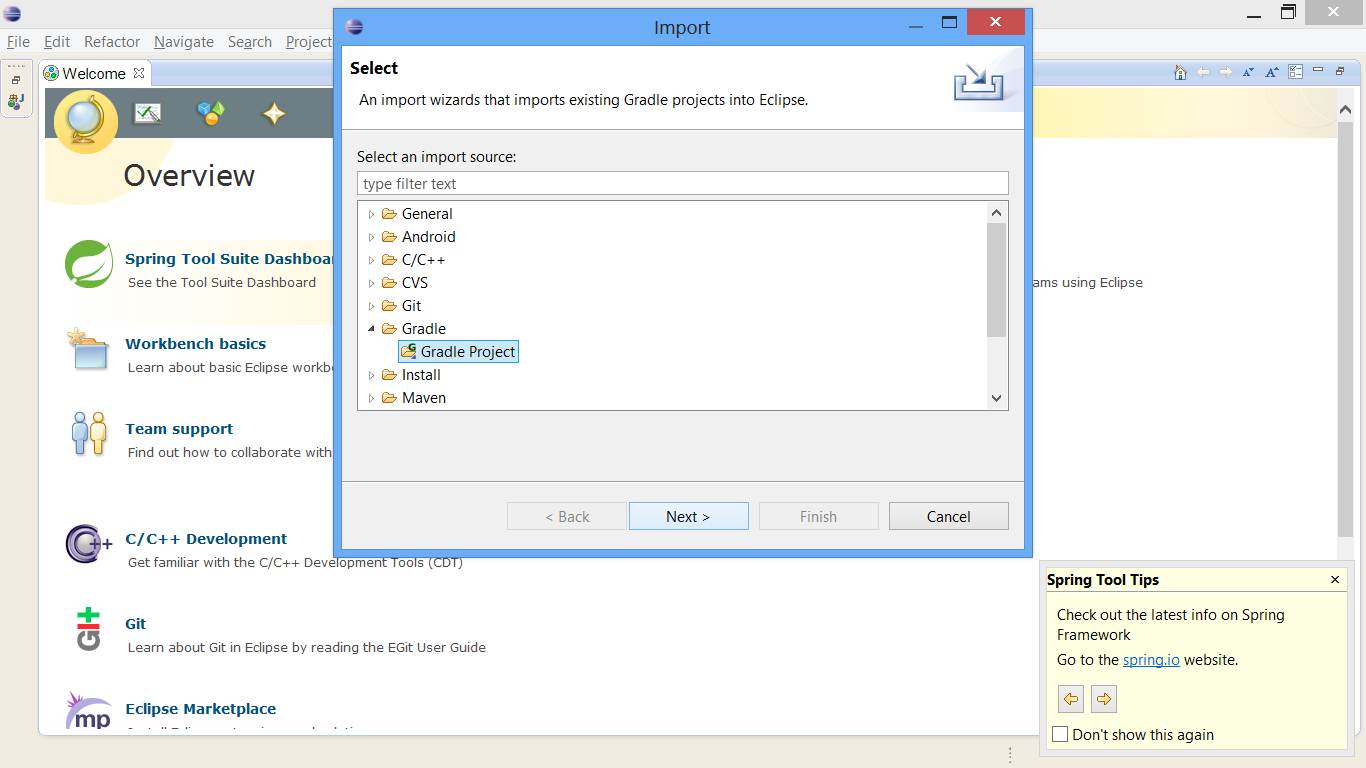


1. Sau 4 phút thì nó mới xong (như hình) thì ta có thể đóng nó lại.

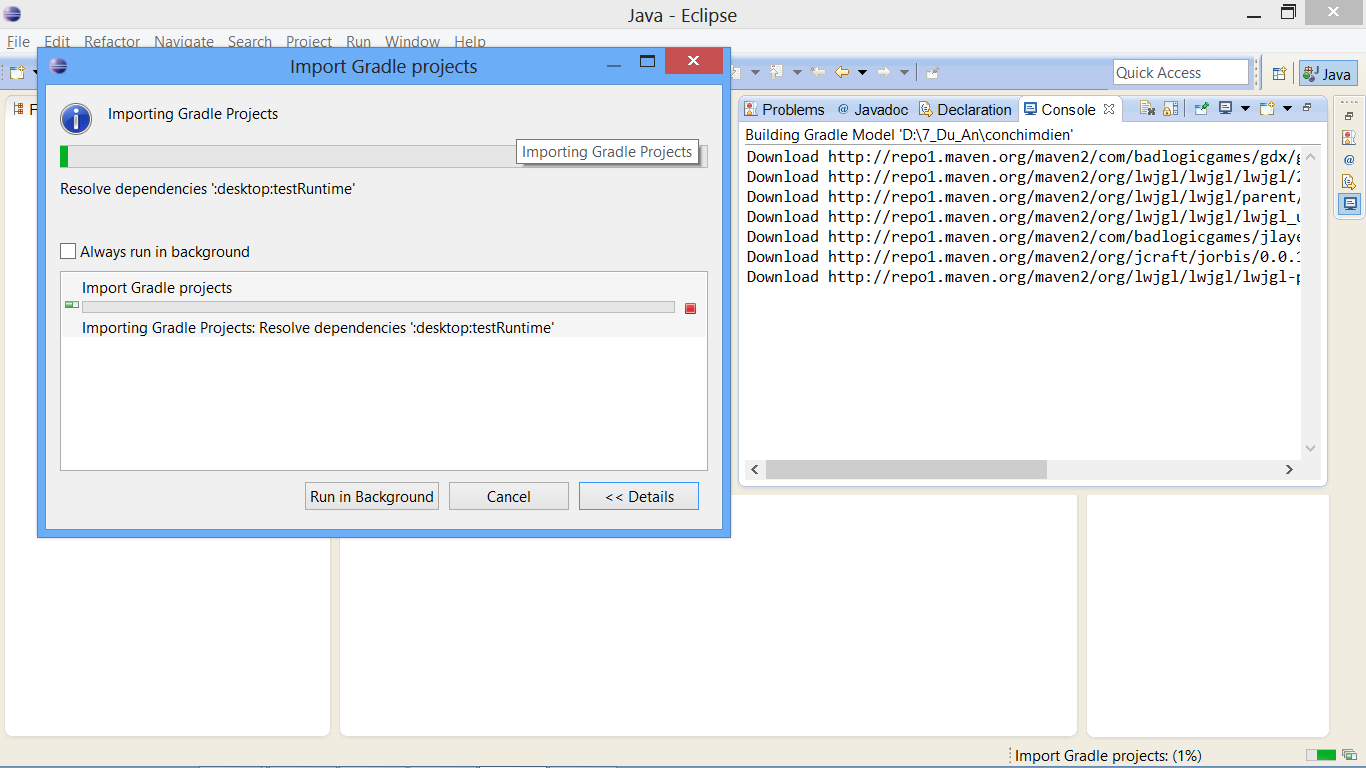


# Nhúng project vào Eclipse

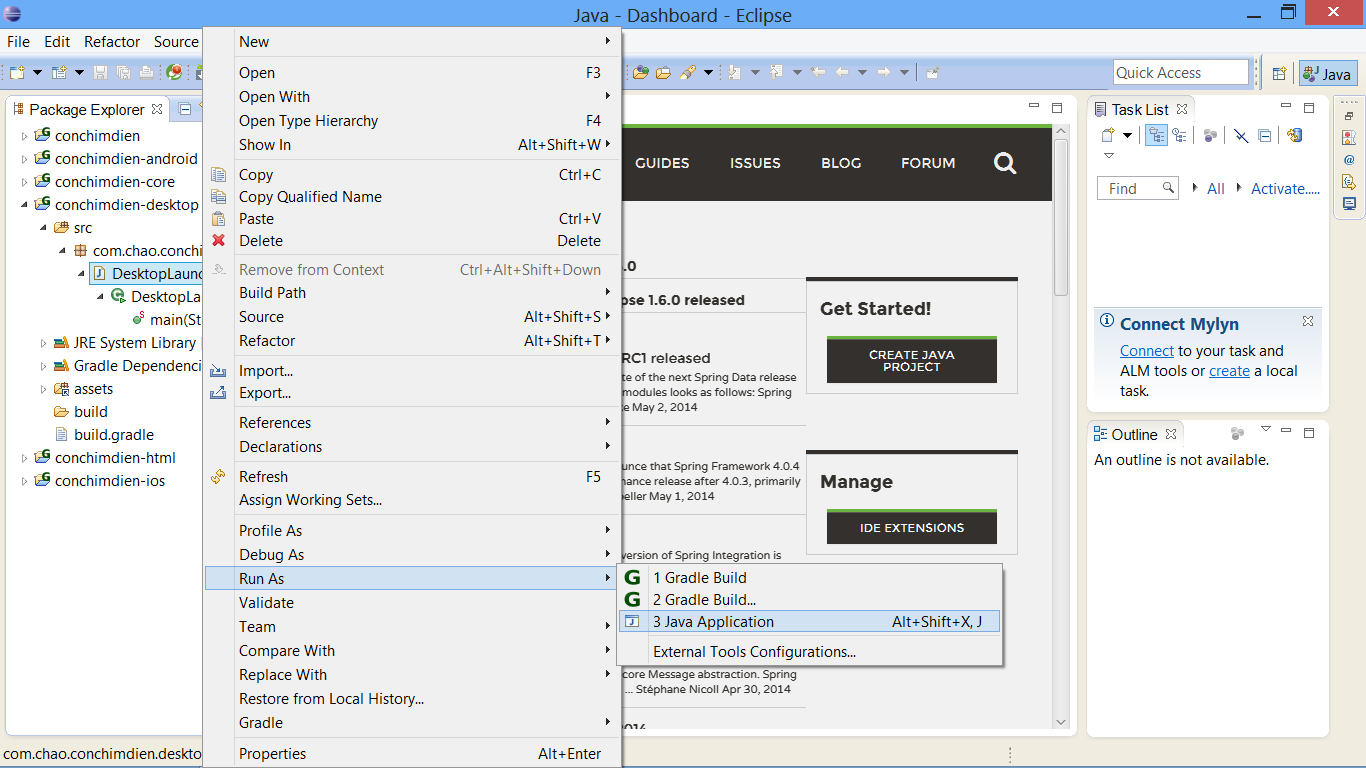
1. Mở Eclipse chọn File -> Import -> Gradle -> Grade Project->Next.



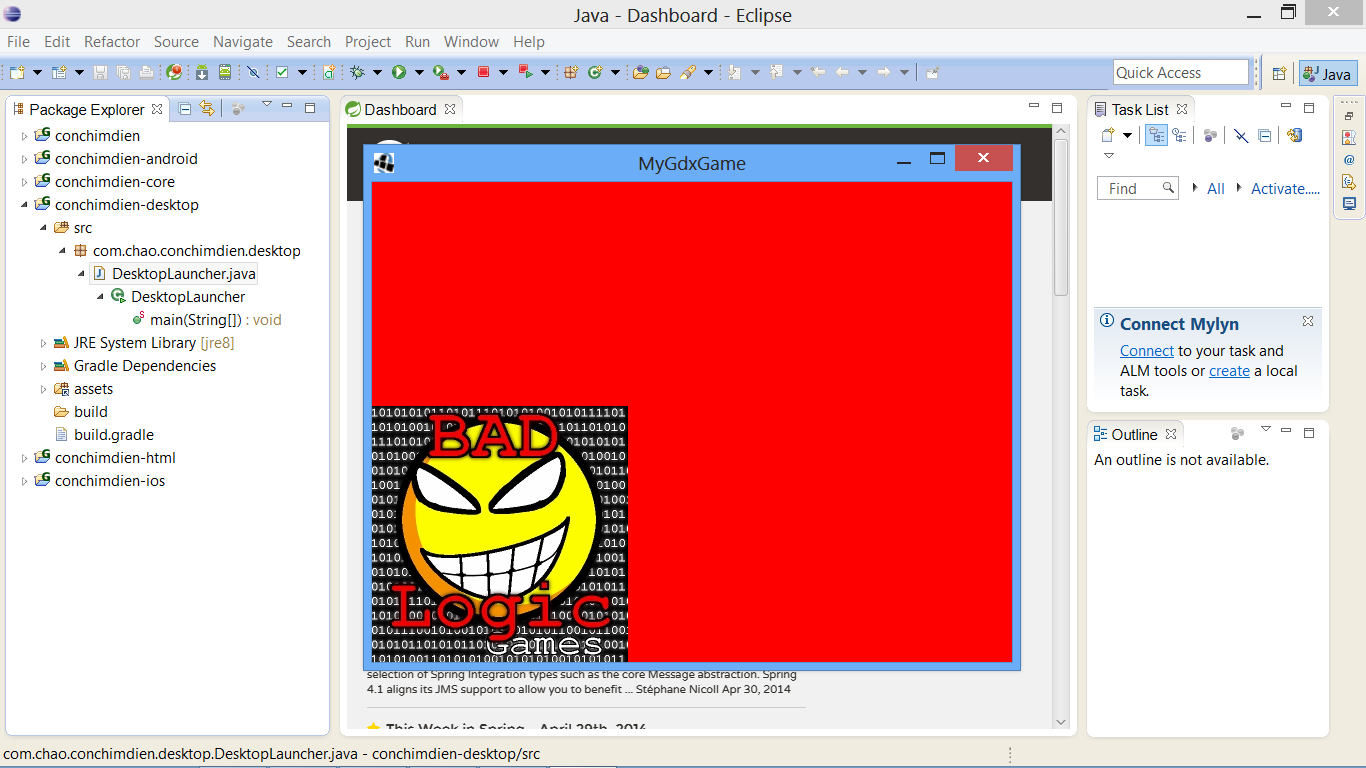
1. Màn hình tiếp theo chọn nút Browser rồi dẫn đến thư mục gốc của project. Sau đó nhấn nút “Build Model”. Eclipse sẽ build và ra giống hình. Ta nhấn vào nút check rồi nhấn Finish. Project Gradle sẽ được nhúng vào eclipse (bước này hơi bị lâu à nhe).



1. Sau khi import xong thì chạy thử project xem cài xong chưa bằng cách chọn project desktop và xuống tới file java, click phải chọn run as-> java application

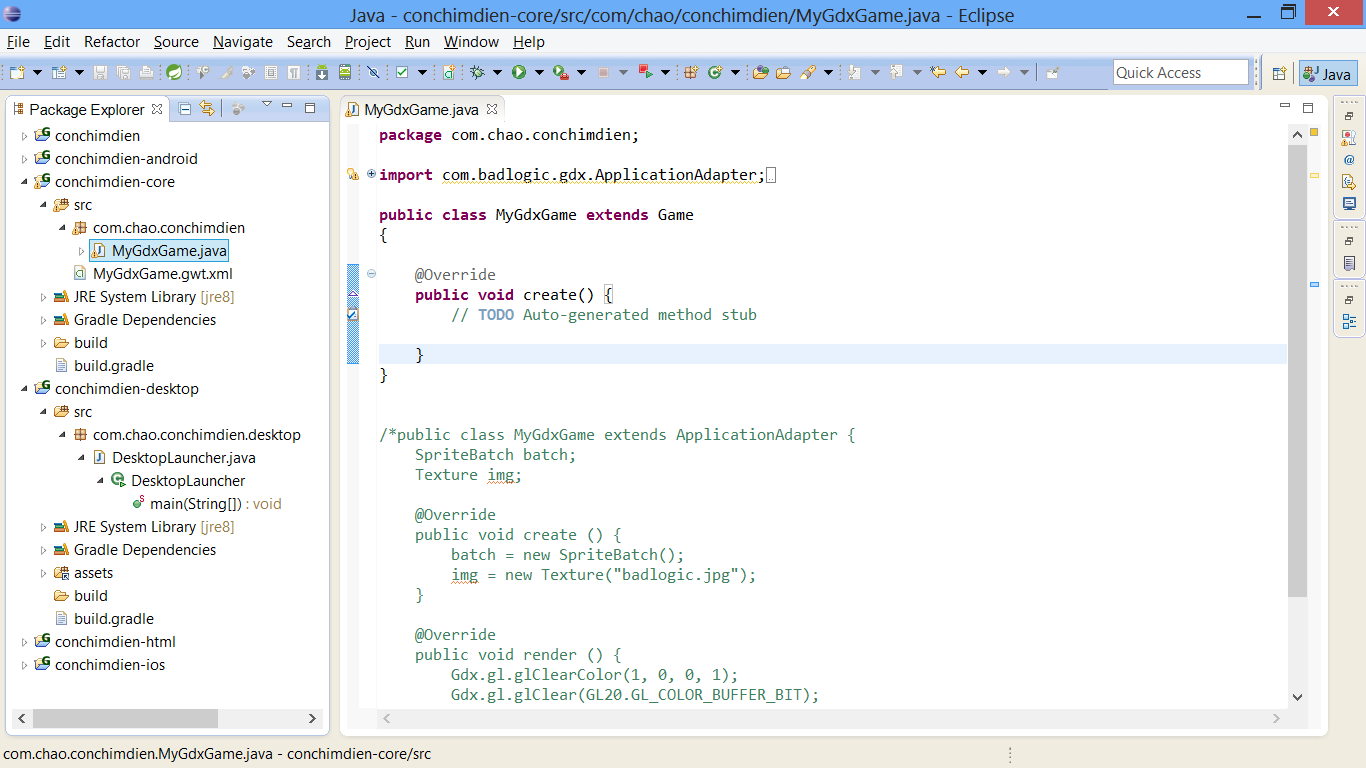


Nếu nó chạy được thì sẽ ra như hình



# Bắt đầu viết mã

1. Mở project “conchimdien-core” (đây là project dùng chung cho các project trên android, ios…) trong src mở file MyGdxGame.java lên, khóa class hiện tại lại, tạo lại một class tên MyGdxGame (trùng tên file java) cho kế thừa từ class Game, nhúng thư viện và “add unimplemented” để ra phương thức onCreate như hình.

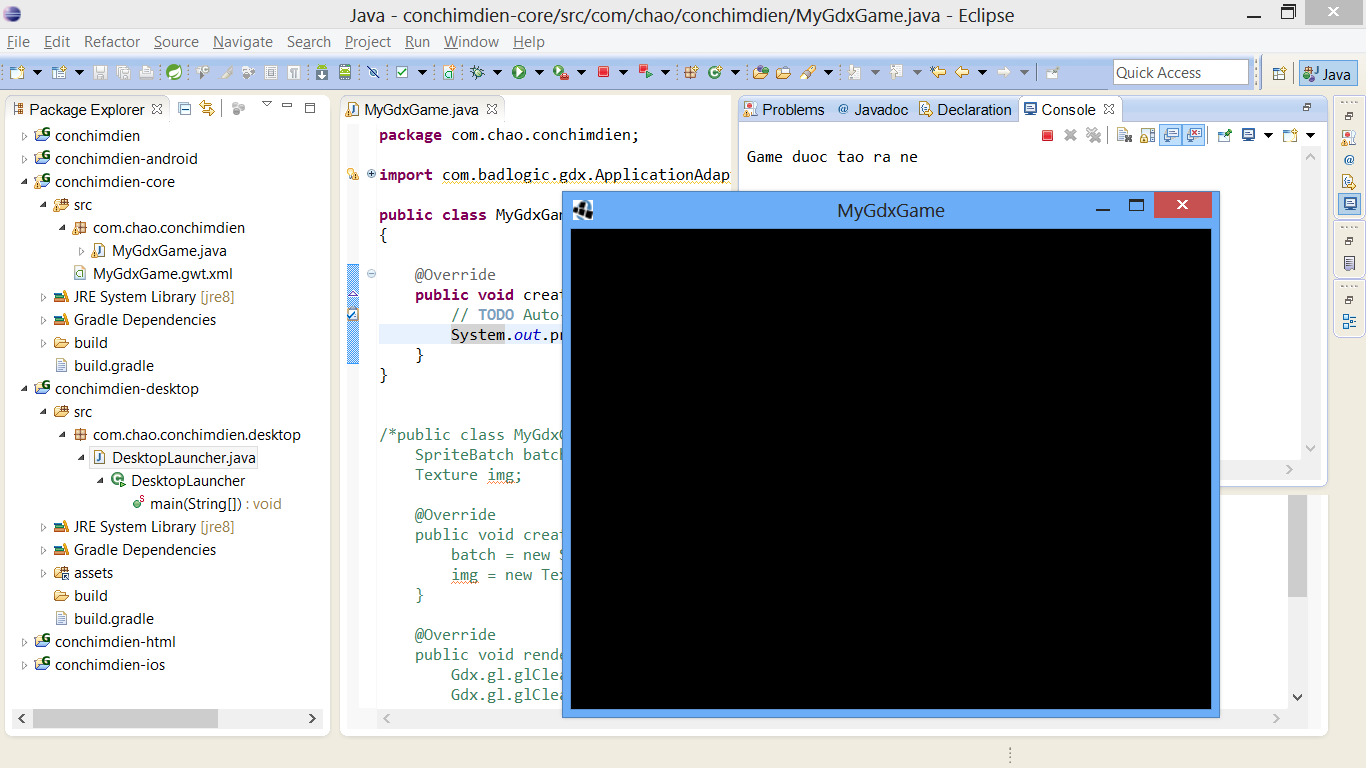


Bằng cách kế thừ từ lớp Game, lớp MyGdxGame trở thành interface giữa mã của ta và các nền tảng khác (android, ios…). Bây giờ bên dưới hậu trường android, ios, html có thể giao tiếp với MyGdxGame.

1. Trong hàm Create viết thêm 1 mã đơn giản như sau:

System.*out*.println("Game duoc tao ra ne");

1. Chạy project desktop (cách chạy đã trình bày ở trên) và ta thấy màn hình đen nhưng trong console hiện thông báo như hình



# Tạo Game Screen (màn hình game)

1. Trong project core tạo một package tên “com.chao.screens
2. Trong package vừa tạo tạo một class mới tên GameScreen thực thi từ giao diện Screen và phát sinh ra một loạt hàm. Trong hàm viết thêm các lệnh print để biết khi chạy, trong render viết thêm 2 lệnh để clear màn hình như đoạn mã bên dưới

**publicclass** GameScreen **implements** Screen{

@Override

**publicvoid** render(**float** delta) {

// **TODO** Auto-generated method stub

//red:10, green:15,blue:230,100% opacity

Gdx.*gl*.glClearColor(10/255.0f,15/255.0f, 230/255.0f,1f);

Gdx.*gl*.glClear(GL20.*GL\_COLOR\_BUFFER\_BIT*);

}

@Override

**publicvoid** resize(**int** width, **int** height) {

// **TODO** Auto-generated method stub

System.*out*.println("man hinh chao - resize");

}

@Override

**publicvoid** show() {

// **TODO** Auto-generated method stub

System.*out*.println("man hinh chao - show");

}

@Override

**publicvoid** hide() {

// **TODO** Auto-generated method stub

System.*out*.println("man hinh chao - hide");

}

@Override

**publicvoid** pause() {

// **TODO** Auto-generated method stub

System.*out*.println("man hinh chao - pause");

}

@Override

**publicvoid** resume() {

// **TODO** Auto-generated method stub

System.*out*.println("man hinh chao - resume");

}

@Override

**publicvoid** dispose() {

// **TODO** Auto-generated method stub

System.*out*.println("man hinh chao - dispose");

}

}

1. Tiếp theo ta gán lớp này vào Class MyGdxGame. Mở lại file MyGdxGame. Trong hàm onCreate viết thêm như sau:

@Override

**publicvoid** create() {

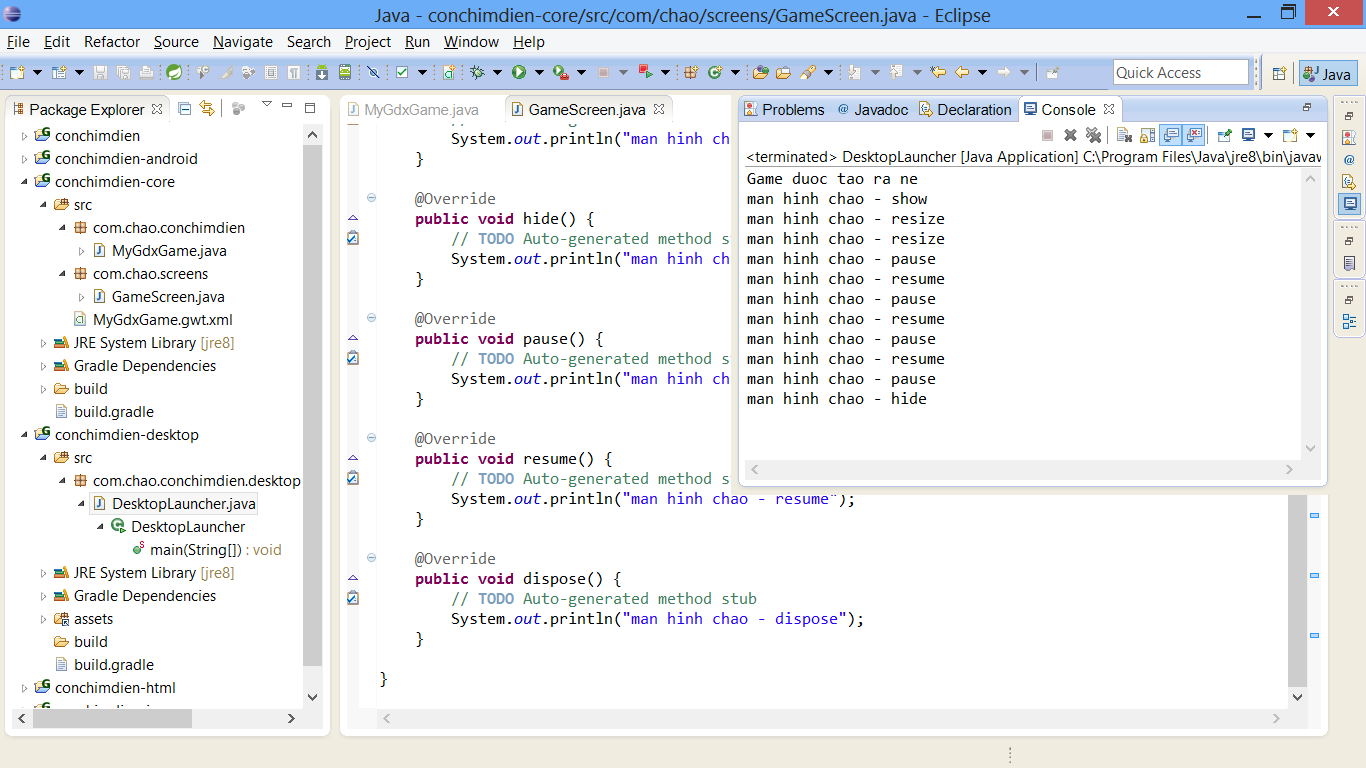
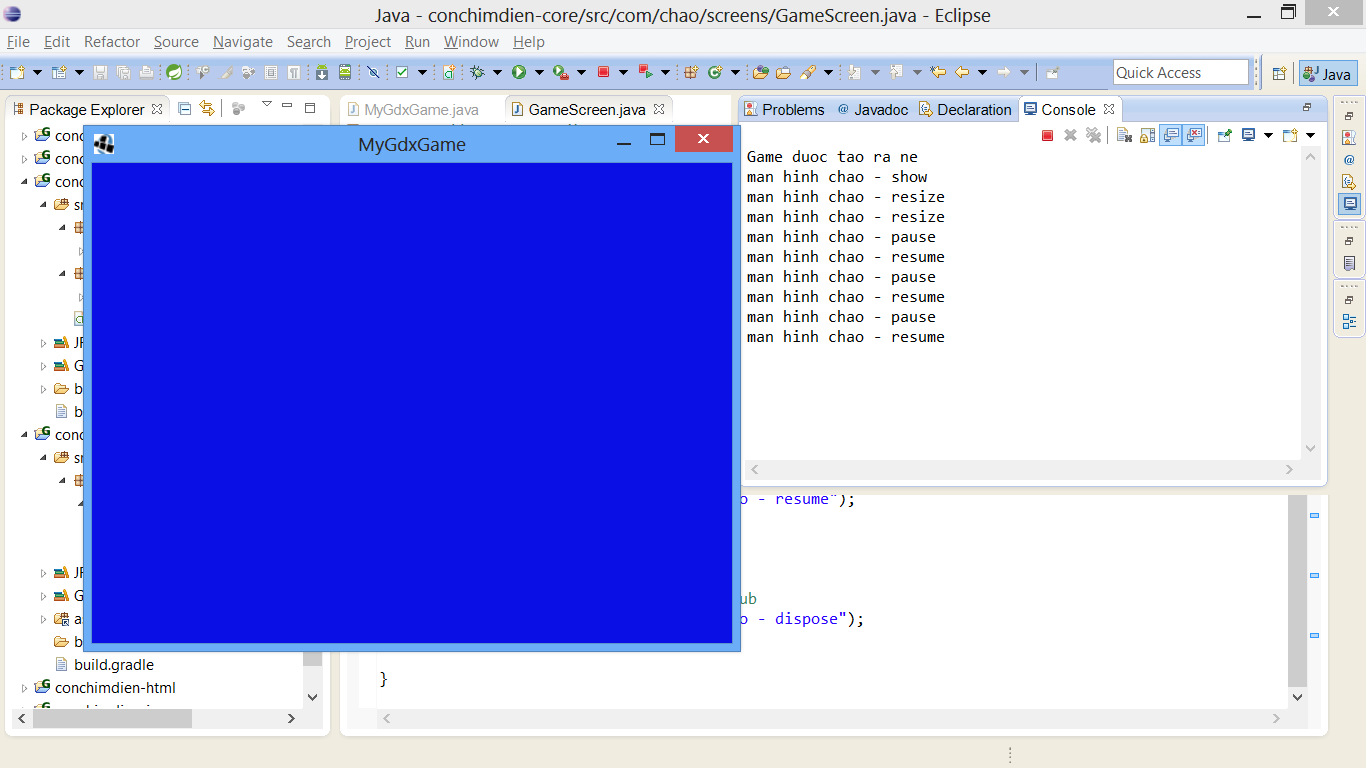
// **TODO** Auto-generated method stub

System.*out*.println("Game duoc tao ra ne");

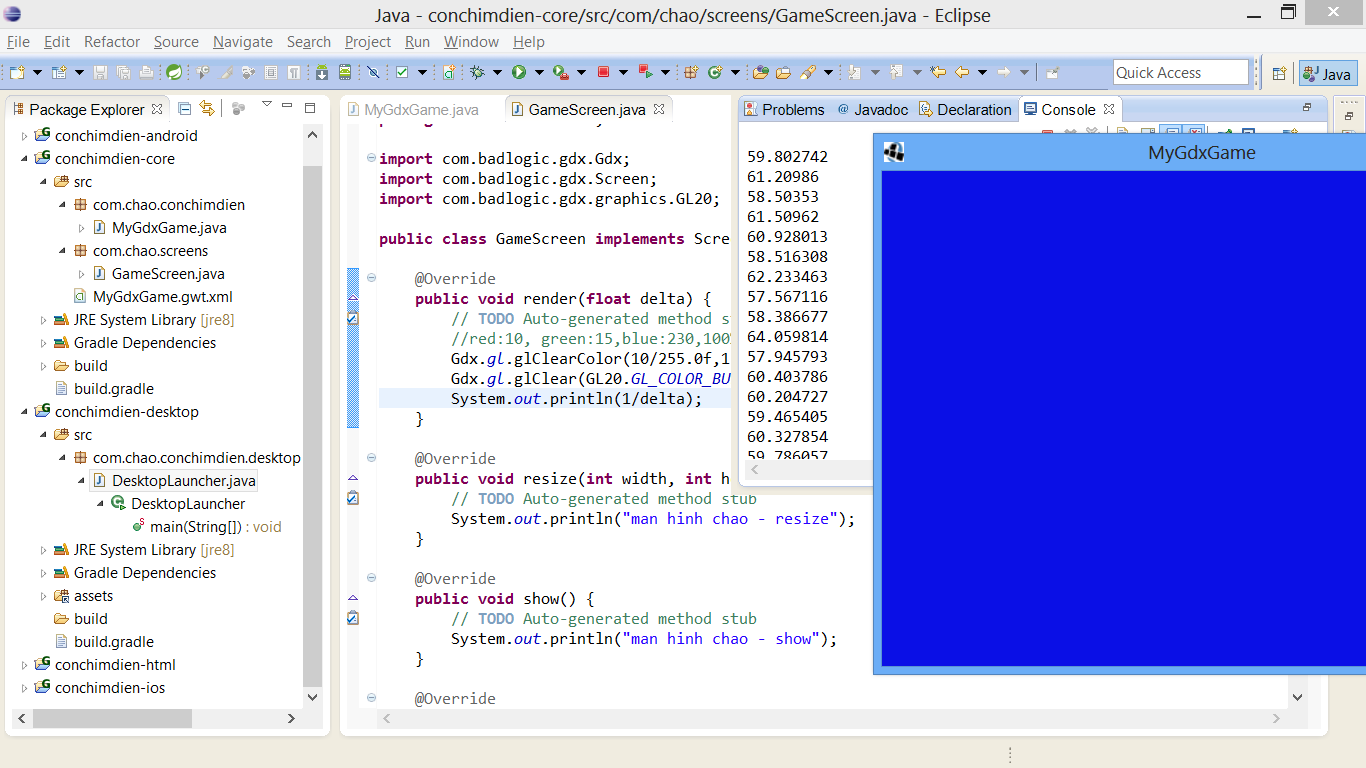
setScreen(**new** GameScreen());

}

1. Chạy project desktop để thấy kết quả, tắt cửa sổ để thấy console phát sinh thêm hide.



1. Mở Class GameScreen, trong hàm render lên và thêm lệnh in ra 1/delta. Chạy desktop và xem kết quả. Biến delta là số giây giữa mỗi lần hàm render được gọi (render được gọi liên tục để vẽ game). Số này thường rất nhỏ, ta lấy 1/delta chính là ra FPS (frame per second) (khung hình trên giây). Tức là một giây sắp sĩ 60 lần gọi hàm render để vẽ lại.



Ta thấy rằng đây chính là vòng lặp game (Game Loop). Trong vòng lặp game ta sẽ làm 2 thứ như sau thứ nhất là cập nhật tất cả các đối tượng game, thứ 2 là render các đối tượng.

GameScreen không nên ôm đồm mọi thứ do đó ta sẽ tạo ra một lớp để phục vụ cập nhật đối tượng và một lớp phục vụ việc render. Sau đó trong GameScreen ta sẽ ủy quyền cập nhật đối tượng và render cho 2 lớp này.

# Tạo GameWorld và GameRenderer

1. Tạo ra một package mới tên “com.chao.gameworld”
2. Trong package mới vừa tạo ta tạo ra 2 class tên “GameWorld” và “GameRenderer”. Trong Class GameWorld viết 1 hàm update nhận vào biến delta và in ra câu đơn giản. Trong class GameRenderer viết hàm render và in ra câu đơn giản. 2 class như sau:

**publicclass** GameWorld {

**publicvoid** update(**float** delta)

{

System.*out*.println("GameWorld - update");

}

}

**publicclass** GameRenderer {

**publicvoid** render()

{

System.*out*.println("GameRenderer - render");

}

}

1. Quay về lớp GameScreen khai báo 2 đối tượng toàn cục của lớp GameWorld và GameRenderer, Viết hàm tạo và khởi tạo 2 đối tượng này. Trong hàm render bỏ hết mã lệnh và gọi hàm update của đối tượng world và render của đối tượng renderer. Mã như sau:

**private** GameWorld world;

**private** GameRenderer renderer;

**public** GameScreen()

{

System.*out*.println("GameScreen - ham tao");

world=**new** GameWorld();

renderer=**new** GameRenderer();

}

@Override

**publicvoid** render(**float** delta) {

// **TODO** Auto-generated method stub

world.update(delta);

renderer.render();

//red:10, green:15,blue:230,100% opacity

/\* Gdx.gl.glClearColor(10/255.0f,15/255.0f, 230/255.0f,1f);

Gdx.gl.glClear(GL20.GL\_COLOR\_BUFFER\_BIT);

System.out.println(1/delta);\*/

}

1. Chạy project desktop để thấy update và render chạy liên tục.

Ta có thể xem qui trình của vận hành và các class theo hình sau:

conchimdien

MyGdxGame: lớp cốt lõi

Các lớp về GamePlay

Các lớp Screen (các view hiển thị)

Framework Helper (hỗ trợ input, image, sound)

GameScreen

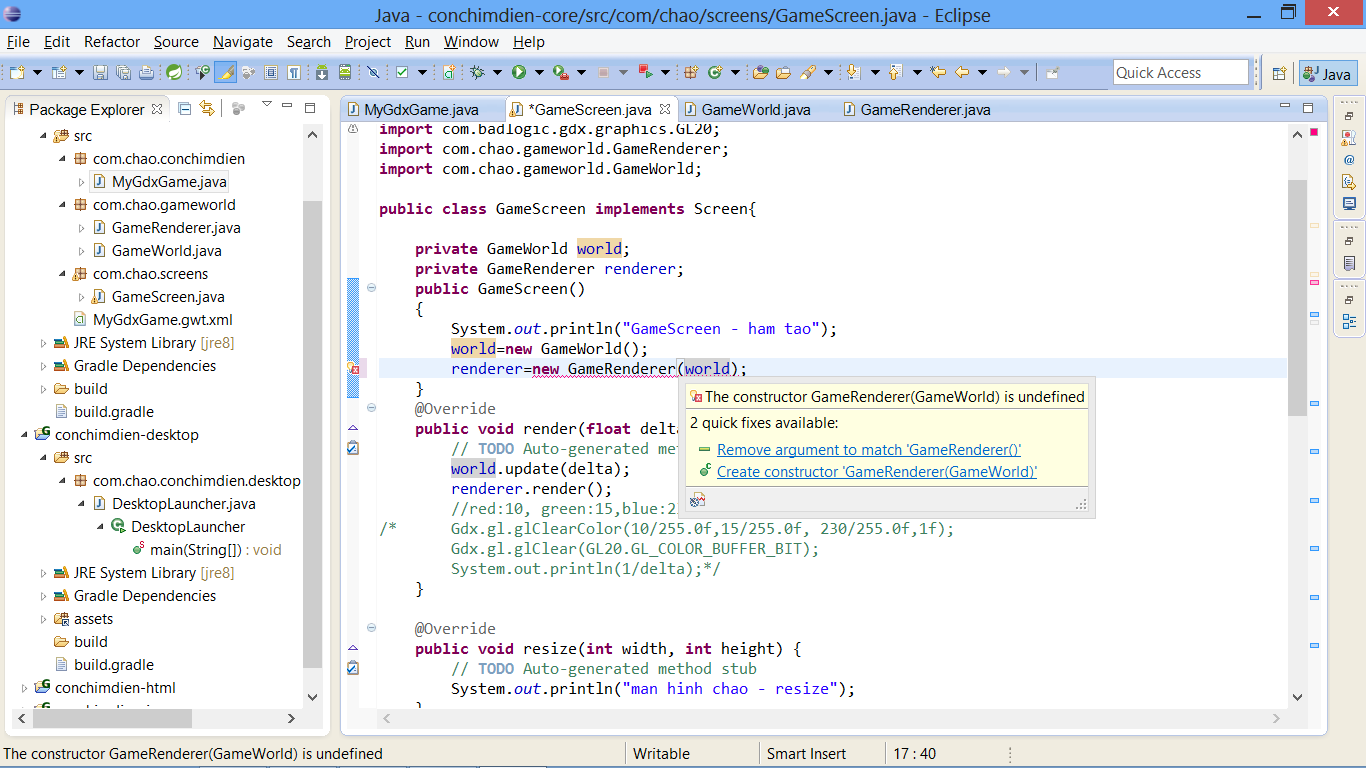
Screen khác

Renderer

World

Các lớp hỗ trợ

1. Trong hàm tạo GameScreen ta sửa lại khi khởi tạo GameRenderer ta truyền world vào luôn. Sau đó đưa vào hiện tooltip vàng ta chọn Create constructor GameRenderer(GameWorld) luôn cho lẹ.



1. Trong class Render khai báo thêm đối tượng myWorld, trong hàm tạo đối tượng world truyền vào ta gán cho myWorld như sau:

**private** GameWorld myWorld;

**public** GameRenderer(GameWorld world) {

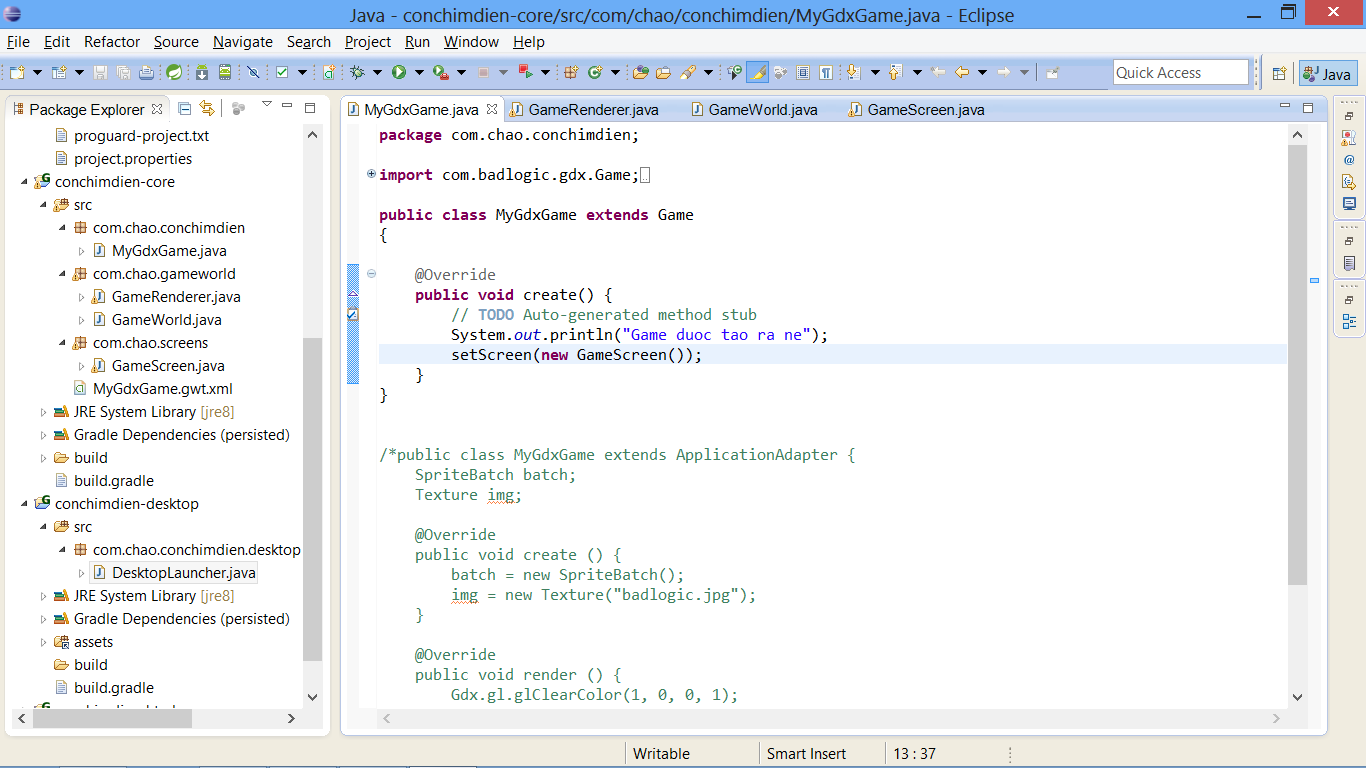
// **TODO** Auto-generated constructor stub

myWorld=world;

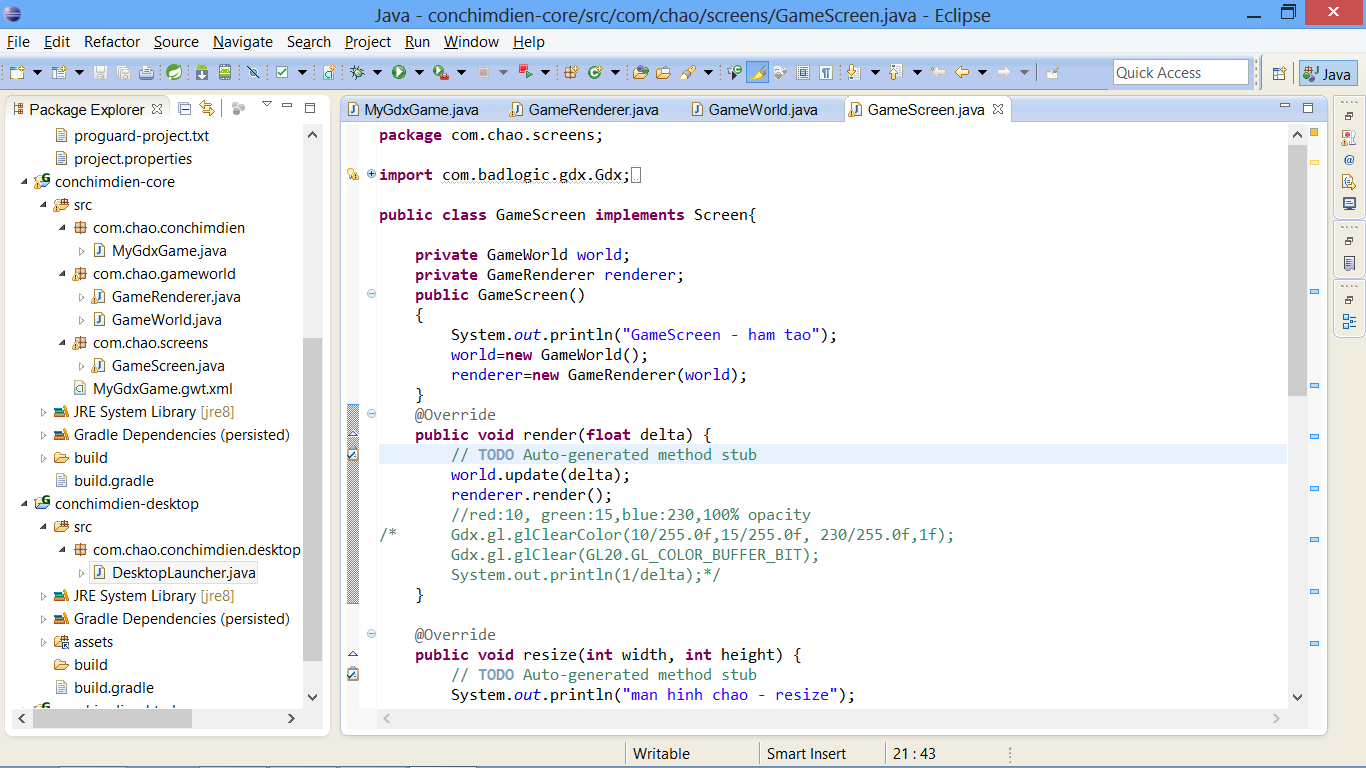
}

# Mối quan hệ

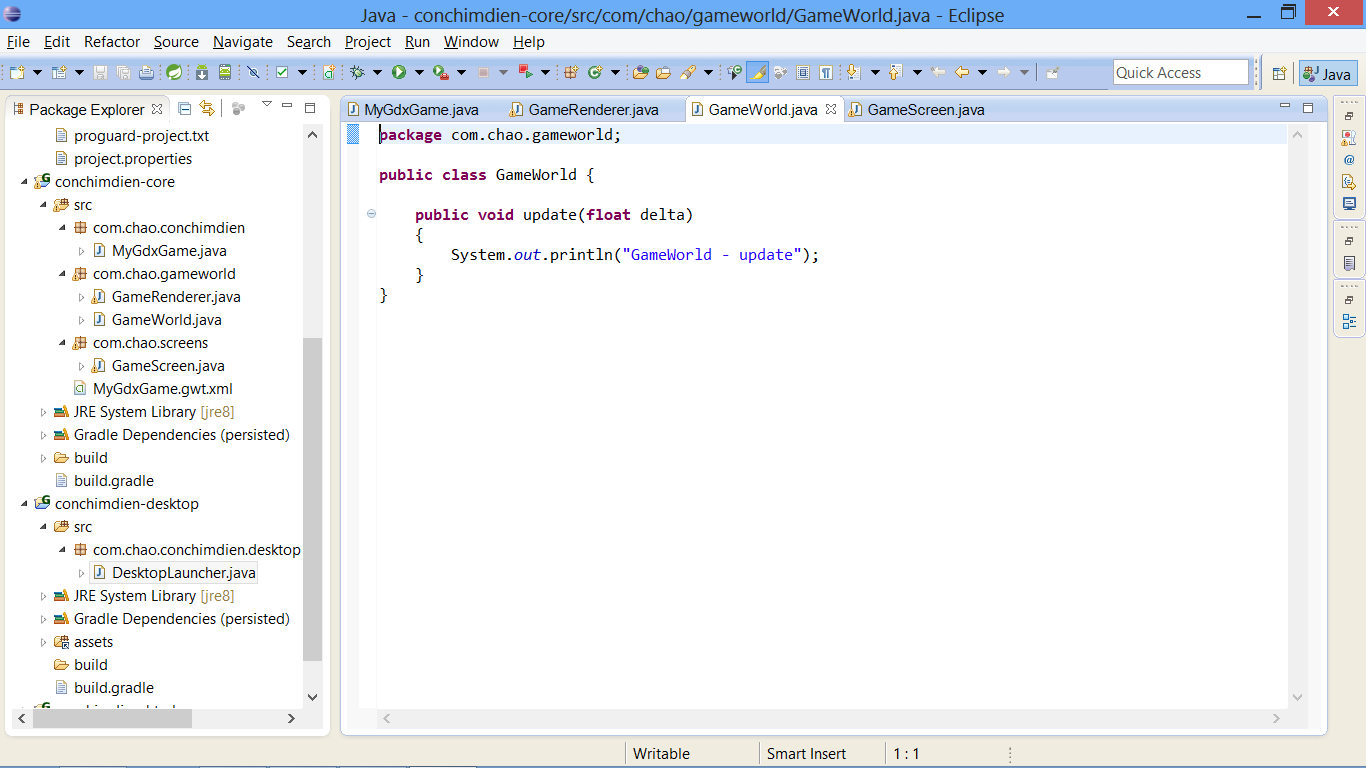
Đến đây ta có thể thấy mối quan hệ giữa 3 class này. Hiểu nôm na MyGdxGame chạy sẽ khởi tạo GameScreen (GameScreen là một màn hình hiển thị game)



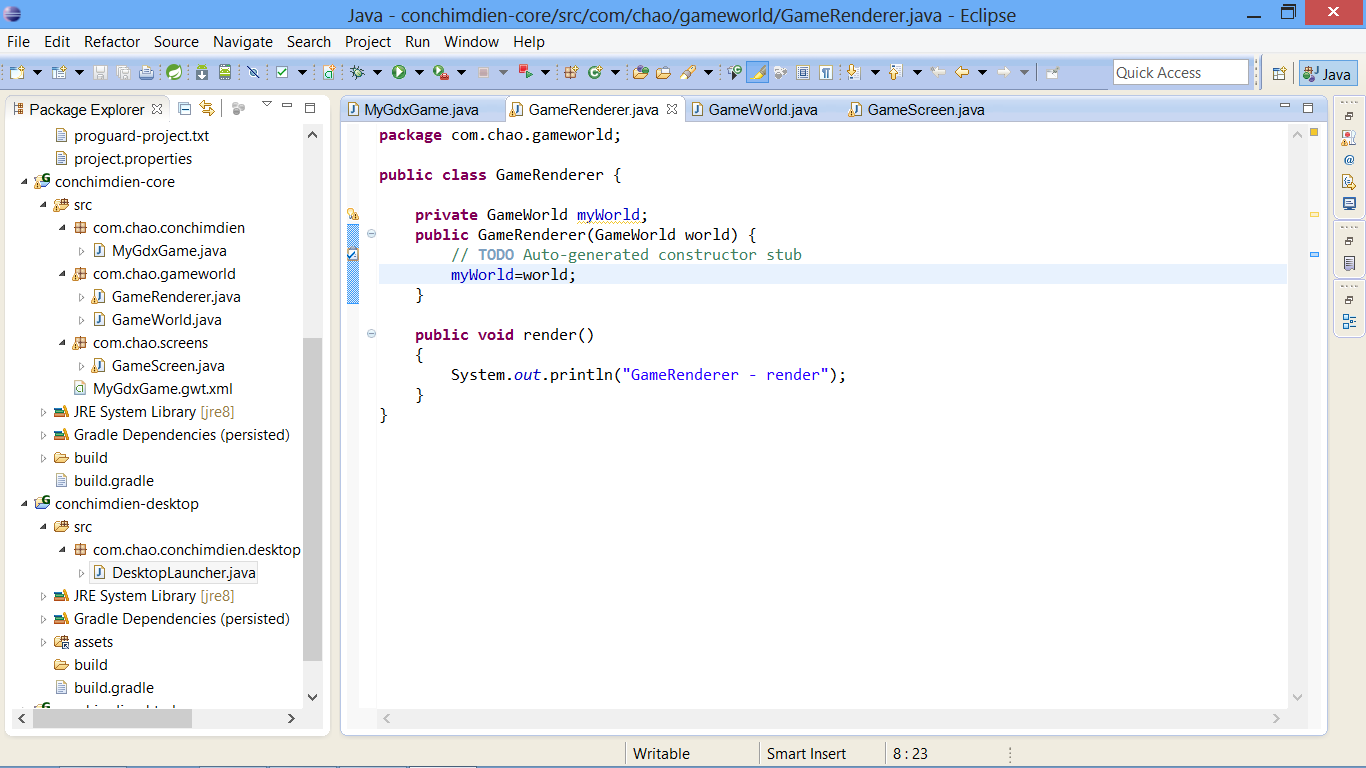
GameScreen (1 màn hình game) sẽ khởi tạo Gameworld và GameRenderer. Và có hàm render, hàm render sẽ cập nhật world rồi gọi render để vẽ lại. Tức là GameWorld nhận nhiệm vụ cập nhật các biến, trạng thái, hình ảnh diễn tiến của game sau đó sẽ dùng dến render để vẽ lại.



GameWorld có hàm update nhận nhiệm vụ cập nhật lại dữ liệu hình ảnh, trạng thái …..



GameRenderer có hàm render nhận nhiệm vụ cập nhật lại game



Khi chạy thấy trong console hiện ra liên tục Gameworld-update và GameRenderer-render, đây là game loop.

# Đổi solution và title

Đây là framework 3D nhưng ta sẽ viết game dạng 2D

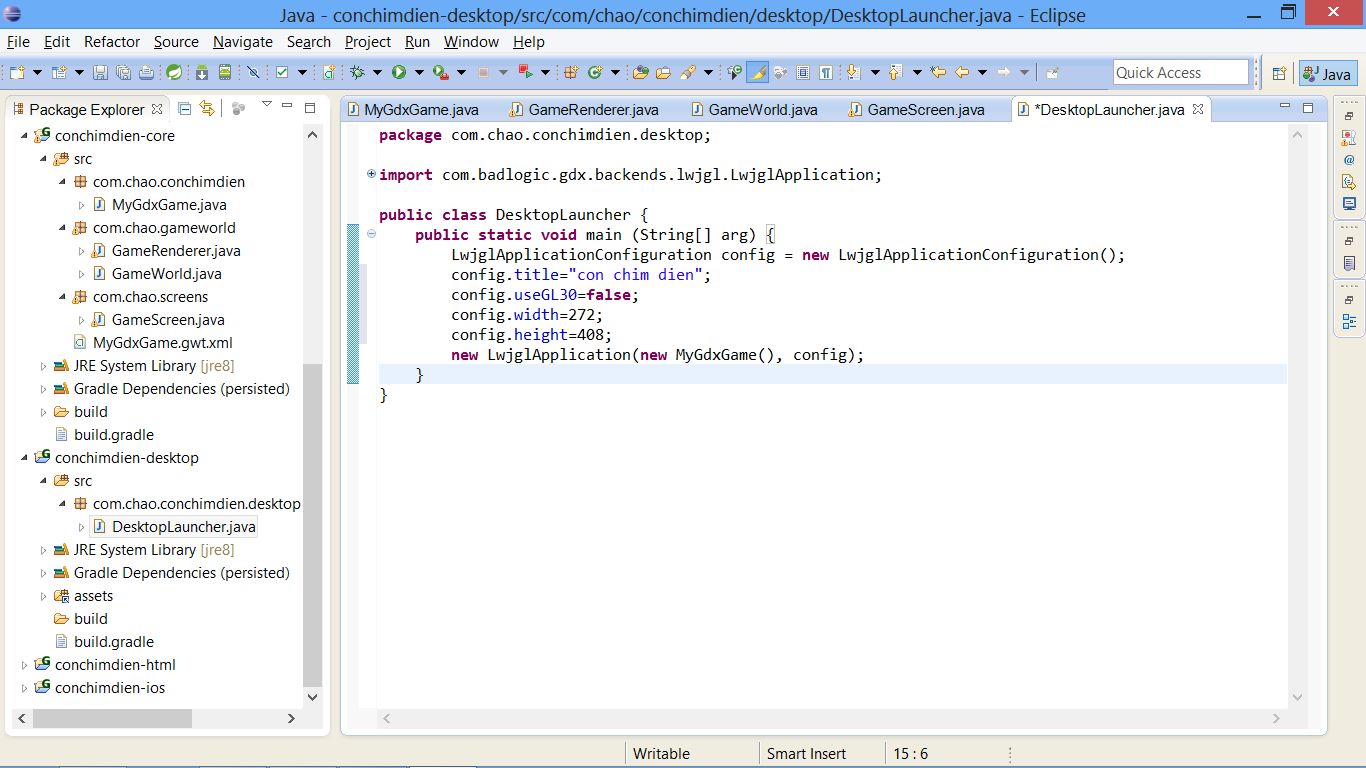
1. Đầu tiên là thay đổi solution, mở file DesktopLauncher.java trong core và

config.title="con chim dien";

config.useGL30=**false**;

config.width=272;

config.height=408;



# Tạo Orthographic Camera và ShapeRenderer

1. Mở file GameRender lên khai báo biến.

private OrthographicCamera cam;

1. Import thư viện.
2. Khởi tạo biến trong hàm tạo như bên dưới . setToOrtho có 3 đối số, : 1. Dùng camera, 2 kích thước chiều rộng, 3 kích thước chiều ngang (ta sẽ đổi nó sau). Nhớ rằng ta dùng resolution 272x408, có nghĩa rằng mọi thứ trong game sẽ được scale 2x khi vẽ ra.

cam =new OrthographicCamera();

cam.setToOrtho(true, 136, 204);

1. Để test camera ta sẽ vẽ ra một hình gì đó và đường viền cho nó. Khai báo 1 biến

private ShapeRenderer shapeRenderer;

1. Import thư viện
2. Khởi tạo shapeRenderer, và gắn nó vào camera trong hàm tạo:

shapeRenderer = new ShapeRenderer();

shapeRenderer.setProjectionMatrix(cam.combined);

Toàn bộ class sẽ như sau:

**publicclass** GameRenderer {

**private** GameWorld myWorld;

**private** OrthographicCamera cam;

**private** ShapeRenderer shapeRenderer;

**public** GameRenderer(GameWorld world) {

// **TODO** Auto-generated constructor stub

myWorld=world;

cam=**new** OrthographicCamera();

cam.setToOrtho(**true**,136,204);

shapeRenderer=**new** ShapeRenderer();

shapeRenderer.setProjectionMatrix(cam.combined);

}

**publicvoid** render()

{

System.*out*.println("GameRenderer - render");

}

}

ShapeRenderer đã sẵn sang, ta sẽ tạo ra một cái gì đó để render, ta sẽ tạo hình chữ nhật cho dễ, có thể tạo nó trong GameRender nhưng không nên, ta nên tạo nó trong GameWorld.

1. Mở GameWorld và sửa như sau:

**publicclass** GameWorld {

**private** Rectangle rect=**new** Rectangle(0, 0, 17, 12);

**public** Rectangle getRect()

{

**return**rect;

}

**publicvoid** update(**float** delta)

{

System.*out*.println("GameWorld - update");

rect.x++;

**if**(rect.x>137)

rect.x=0;

}

}

1. Mở file GameRender và sửa lại hàm render như sau (giải thích trong code):

**publicvoid** render() {

System.*out*.println("GameRenderer - render");

/\* 1. ve background den. nógiúpchốngnhấpnháy \*/

Gdx.*gl*.glClearColor(0, 0, 0, 1);

Gdx.*gl*.glClear(GL20.*GL\_COLOR\_BUFFER\_BIT*);

/\* 2. Bat dauvehinhchunhat\*/

// nóvới ShapeRenderer bắtđầuvẽhình (hìnhtôdầybêntrong)

shapeRenderer.begin(ShapeType.*Filled*);

// chon RGB Color : 87, 109, 120 và opacity khongmờ

shapeRenderer.setColor(87 / 255.0f, 109 / 255.0f, 120 / 255.0f, 1);

// vehìnhchữnhậttừ myWorld, dùng ShapeType.Filled

shapeRenderer.rect(myWorld.getRect().x, myWorld.getRect().y,

myWorld.getRect().width, myWorld.getRect().height);

// noivoi shapeRenderer ketthucviecve

// can phailamdieu nay mỗilầnchạy

shapeRenderer.end();

/\* 3. vẽđườngviềnchohìnhchữnhật \*/

shapeRenderer.begin(ShapeType.*Line*);

shapeRenderer.setColor(255 / 255.0f, 109 / 255.0f, 120 / 255.0f, 1);

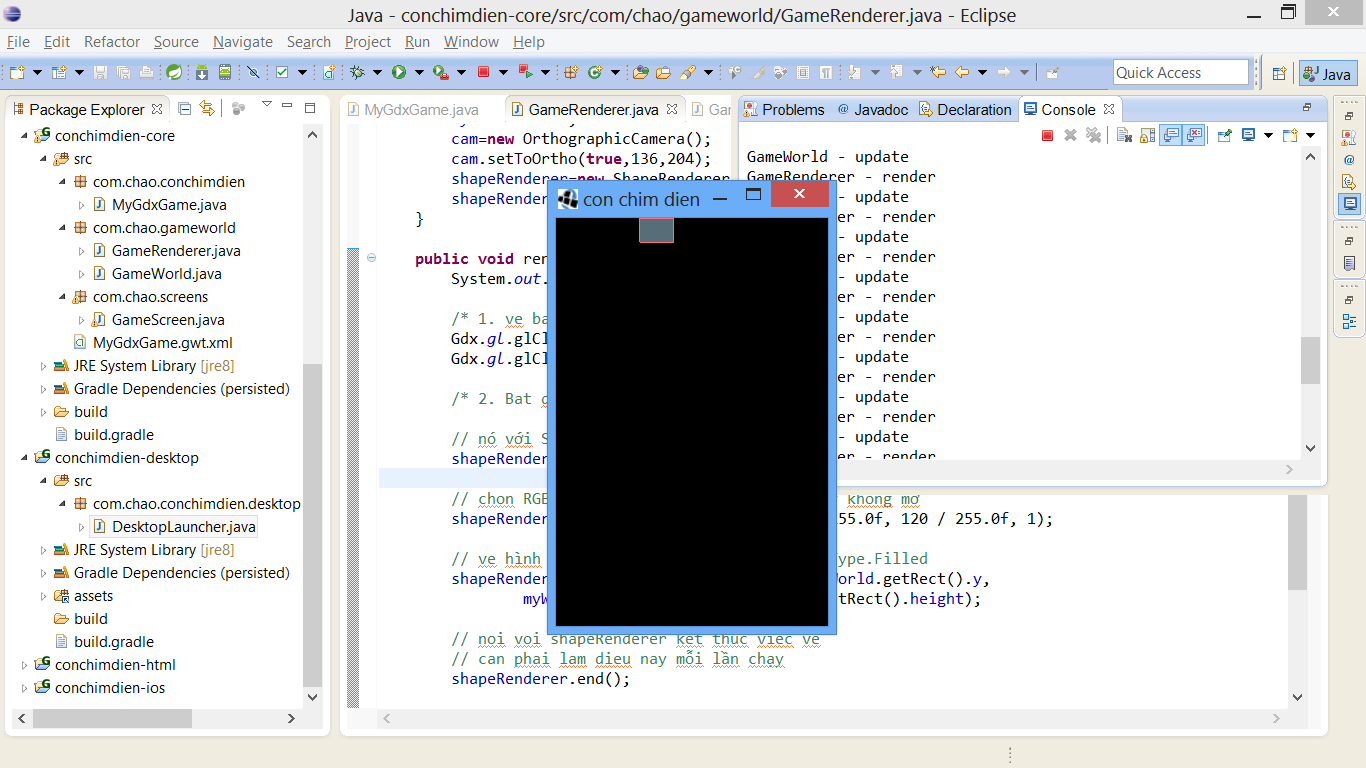
shapeRenderer.rect(myWorld.getRect().x, myWorld.getRect().y,

myWorld.getRect().width, myWorld.getRect().height);// dung ShapeType.Line

shapeRenderer.end();

}

1. Chạy DesktopLauncher xem kết quả ta sẽ thấy một hình chữ nhật có border chạy từ trái sang phải, khi chạy hết sẽ quay lại và chạy lại.



Đến đây ta thấy rằng ta có thể vẽ hình và di chuyển thì có thể vẽ ảnh và di chuyển cũng tương tự.

# Tạo đối tượng Bird

1. Trong project core tạo một package mới tên com.chao.GameObject (package này sẽ chứa các đối tượng cho game).
2. Trong package vừa tạo tạo một class mới tên Bird.java.
3. Trong class vừa tạo ta tạo ra các biến gồm position (lưu vị trí), velocity (lưu vận tốc), và acceleration (lưu gia tốc). Ngoài ra còn cần biến rotation (lưu góc quay), và chiều rộng chiều cao (width, height).

**publicclass** Bird {

**private** Vector2 position;

**private** Vector2 velocity;

**private** Vector2 acceleration;

**privatefloat**rotation;

**privateint**width;

**privateint**height;

**public** Bird(**float** x, **float** y, **int** width, **int** height)

{

**this**.width=width;

**this**.height=height;

position=**new** Vector2(x, y);

velocity=**new** Vector2(0, 0);

acceleration=**new** Vector2(0, 460);

}

}

Class Vector2 là một class mạnh trong libGDX gồm có 2 trục là x và y.

1. Viết thêm các hàm để hỗ trợ việc lấy chiều rộng chiều cao …. ra

**publicfloat** getX() {

**return**position.x;

}

**publicfloat** getY() {

**return**position.y;

}

**publicfloat** getWidth() {

**return**width;

}

**publicfloat** getHeight() {

**return**height;

}

**publicfloat** getRotation() {

**return**rotation;

}

1. Xây thêm hàm update như sau:

**publicvoid** update(**float** delta)

{

velocity.add(acceleration.cpy().scl(delta));

**if**(velocity.y>200)

{

velocity.y=200;

}

position.add(velocity.cpy().scl(delta));

}

1. Xây tiếp 1 hàm tên onClick như sau:

**publicvoid** onClick()

{

velocity.y=-140;

}

1. Mở lại file GameWorld bỏ phần vẽ và khởi tạo con chim như sau:

**publicclass** GameWorld {

**private** Bird bird;

**public** GameWorld(**int** midPointY)

{

bird=**new** Bird(33, midPointY-5, 17, 12);

}

**publicvoid** update(**float** delta)

{

bird.update(delta);

}

**public** Bird getBird()

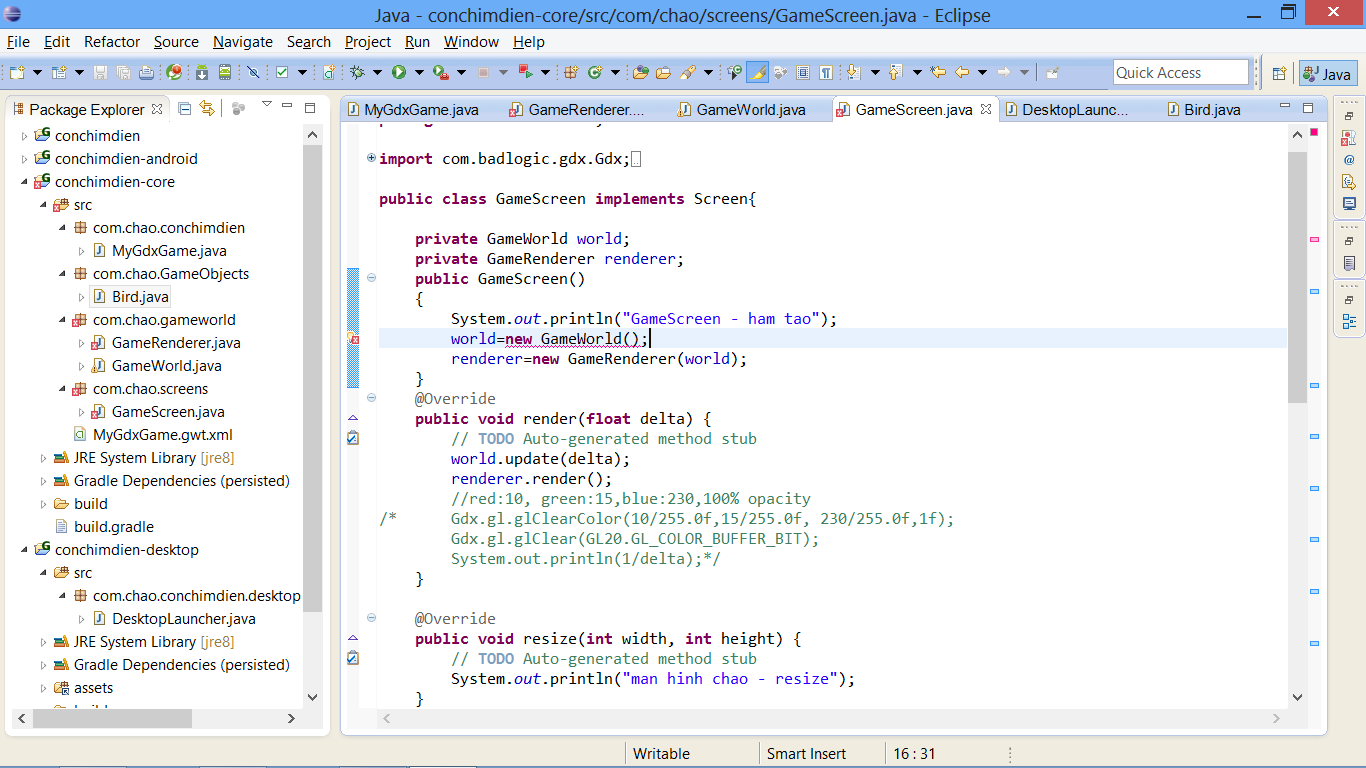
{

**return**bird;

}

}

1. Khi đó GameScreen sẽ bị lỗi như hình



Ta sửa lại hàm GameScreen như sau:

**public** GameScreen() {

System.*out*.println("GameScreen - ham tao");

**float** screenWidth = Gdx.*graphics*.getWidth();

**float** screenHeight = Gdx.*graphics*.getHeight();

**float** gameWidth = 136;

**float** gameHeight = screenHeight / (screenWidth / gameWidth);

**int** midPointY = (**int**) (gameHeight / 2);

world = **new** GameWorld(midPointY);

renderer = **new** GameRenderer(world);

}

1. File GameRender cũng còn bị lỗi do gọi vẽ hình chữ nhật, mở file GameRender.java lên, trong hàm render bỏ bớt phần 2 và 3 (các phần vẽ hình chữ nhật).

# Tạo phần framework helper

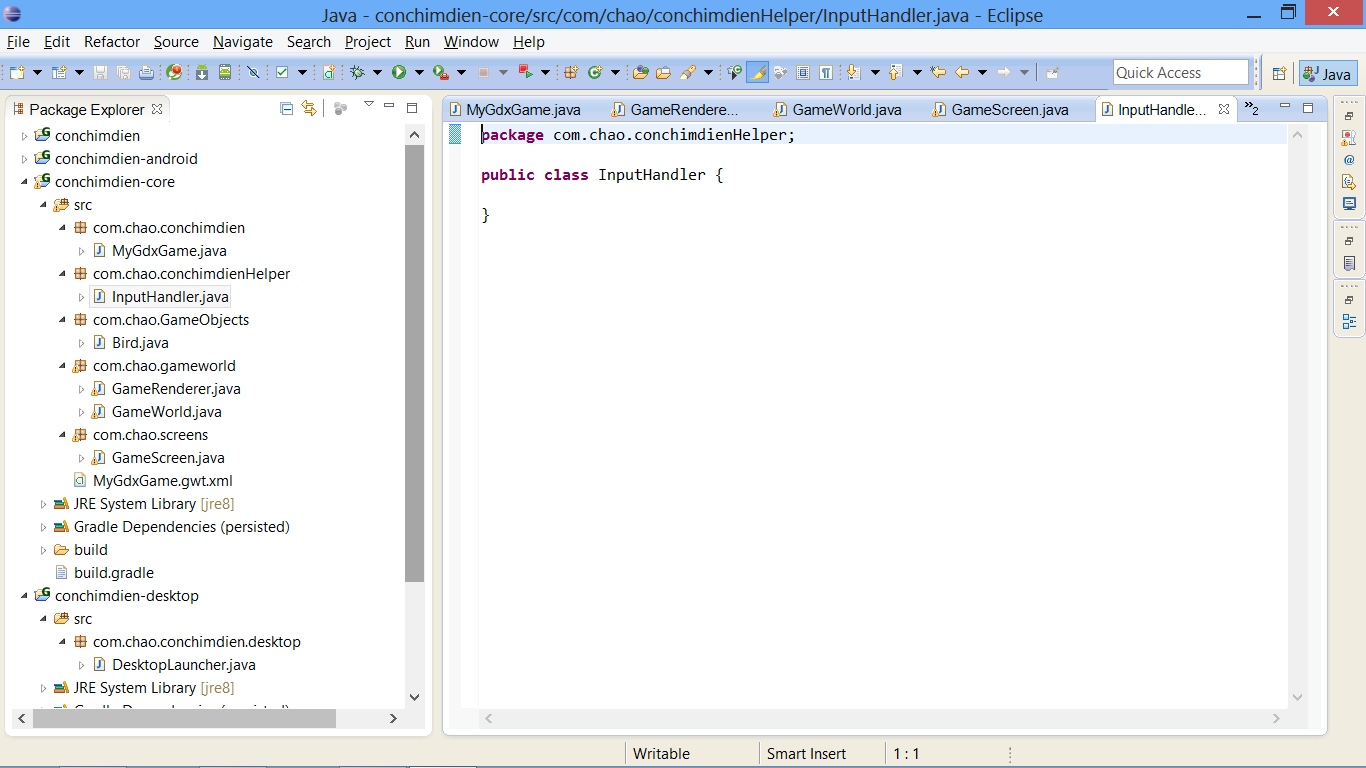
Chúng ta đã tạo ra con chim giờ ta phải điều khiển nó do đó ta thêm một input handler.

Nhìn lại sơ đồ ở bước 9

|  |  |
| --- | --- |
| Ta thấy ở cấp 3 có phần Framework Helper, đây là nơi ta điều khiển đầu vào (input) cũng như nạp hình ảnh, âm thanh.  Ở đây ta sẽ tạo ra 2 class.  Class 1 có thể đặt tên là “InputHandler” dùng để xử lý các điều khiển đầu vào (input).  Class 2 tên “AssetLoader” để nạp hình ảnh, âm thanh…. |  |

# Tạo class “InputHandler”

1. Trong project core ta tạo tiếp một package mới tên “com.chao.conchimdienHelper”. Trong packager này ta tạo tiếp một class mới tên InputHandler như hình



1. Mở class InputHandler vừa tạo lên cho nó thực thi từ InputProcesor và add unimplemented method ta được một đống hàm như mã bên dưới.

**publicclass** InputHandler **implements** InputProcessor{

@Override

**publicboolean** keyDown(**int** keycode) {

// **TODO** Auto-generated method stub

**returnfalse**;

}

@Override

**publicboolean** keyUp(**int** keycode) {

// **TODO** Auto-generated method stub

**returnfalse**;

}

@Override

**publicboolean** keyTyped(**char** character) {

// **TODO** Auto-generated method stub

**returnfalse**;

}

@Override

**publicboolean** touchDown(**int** screenX, **int** screenY, **int** pointer, **int** button) {

// **TODO** Auto-generated method stub

**returnfalse**;

}

@Override

**publicboolean** touchUp(**int** screenX, **int** screenY, **int** pointer, **int** button) {

// **TODO** Auto-generated method stub

**returnfalse**;

}

@Override

**publicboolean** touchDragged(**int** screenX, **int** screenY, **int** pointer) {

// **TODO** Auto-generated method stub

**returnfalse**;

}

@Override

**publicboolean** mouseMoved(**int** screenX, **int** screenY) {

// **TODO** Auto-generated method stub

**returnfalse**;

}

@Override

**publicboolean** scrolled(**int** amount) {

// **TODO** Auto-generated method stub

**returnfalse**;

}

}

Trong các hàm này ta chỉ quan tâm đến hàm touchDown, tức là hàm khi chạm ngón tay vào.

Khi hàm này gọi ta sẽ gọi tới hàm onClick của con chim. Tuy nhiên ở đây lại không có chim để click, do đó ta phải yêu cầu gởi chim qua. GameScreen gởi chim qua để ta click. GameWorld đang nắm giữ chim, và nó có quan hệ với GameScreen do đó ta sẽ nhờ GameScreen gởi chim qua cho InputHandler để InputHandler có thể gọi onClick của chim.

1. Trong class InputHandler ta khai báo thêm biến Bird và viết hàm tạo để nhận Bird từ ngoài truyền vào như sau:

**private** Bird myBird;

**public** InputHandler(Bird bird)

{

**this**.myBird=bird;

}

1. Trong hàm touchDown ta thêm để gọi click của Bird như sau:

@Override

**publicboolean** touchDown(**int** screenX, **int** screenY, **int** pointer, **int** button) {

// **TODO** Auto-generated method stub

myBird.onClick();

**returnfalse**;

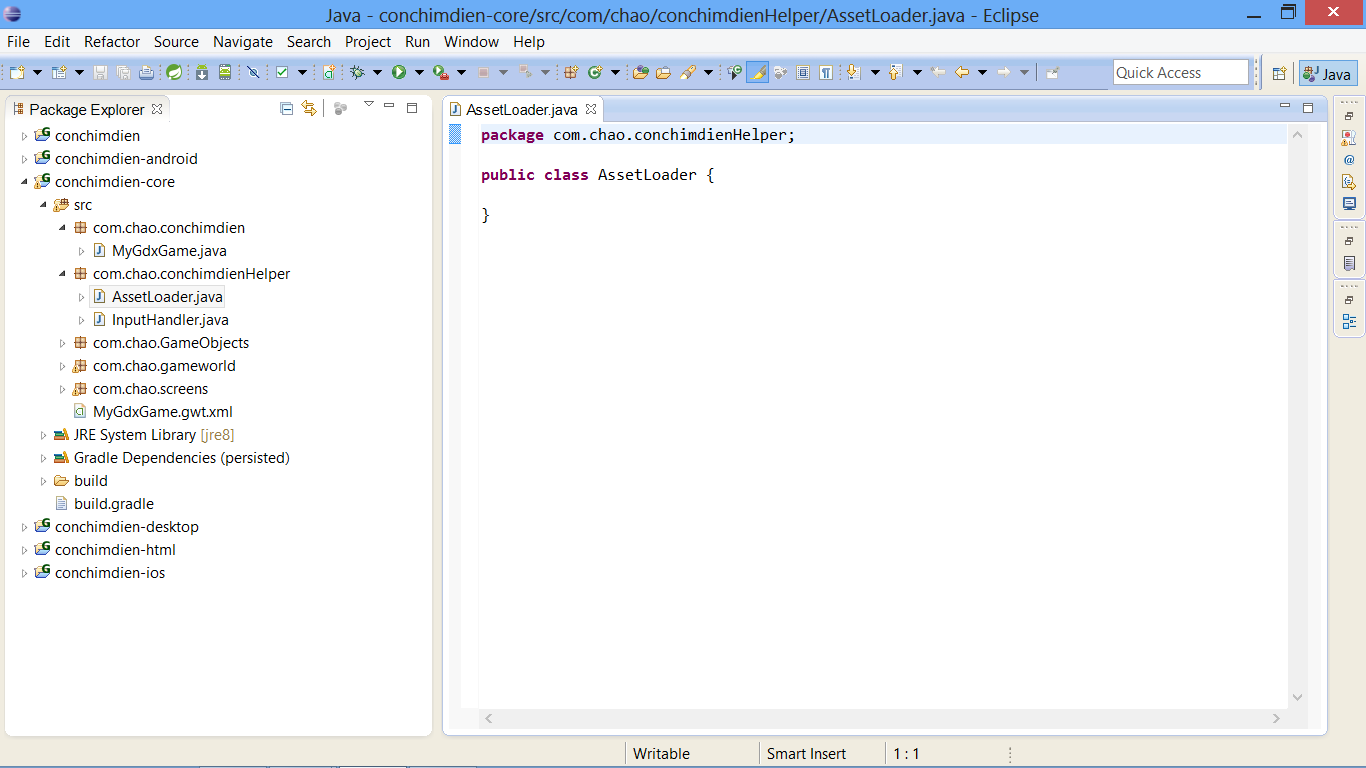
}

1. Quay lại GameScreen.java. Bổ sung lệnh như sau vào cuối của hàm tạo GameScreen, lệnh này sẽ đưa điều khiển vào cho game thông qua các hàm trong InputHandler, ngoài ra nó còn cần phải truyền vào chim lấy từ world ra.

Gdx.*input*.setInputProcessor(**new** InputHandler(world.getBird()));

# AssetLoader

1. Cũng trong package com.chao.conchimdien ta tạo thêm một class tên AssetLoader như hình bên dưới. Class này dùng để nạp Textture.



Texture: ta có thể hình dung đây là một file hình ảnh lớn. Chúng ta sẽ bỏ tất cả các tấm hình nhỏ của game vào hình này sau đó sẽ cắt ra từng vùng để hiển thị.

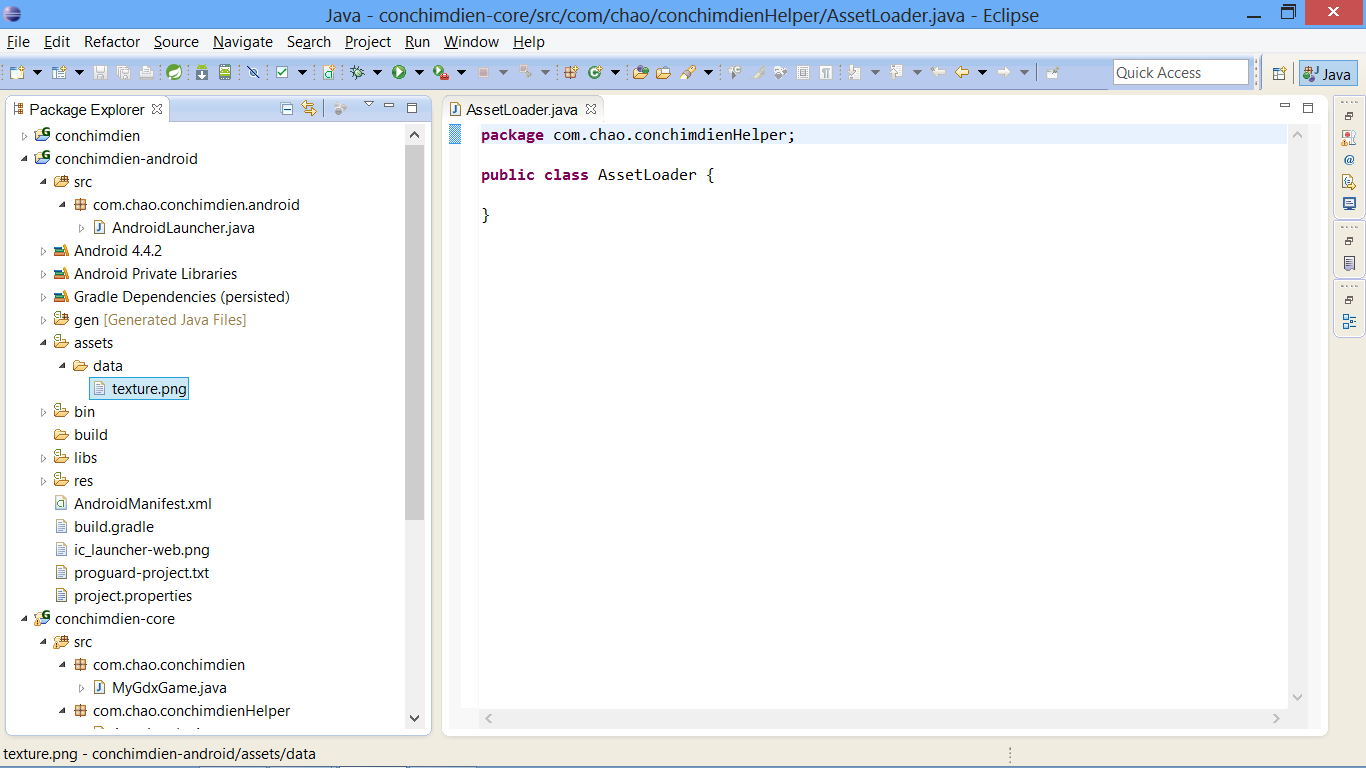
TextureRegion: là một hình chữ nhật ta cắt ra từ texture (do texture là 1 hình lớn có nhiều hình con bên trong).

Animation: Ta có thể có nhiều texture region và tạo ra một sự chuyển động dựa vào việc thay đổi các hình texture region.

Nhìn hình bên dưới, ta thấy toàn bộ hình là một texture, trong đó ta sẽ cắt ra từng hình chữ nhật là từng Texture Region như thành phố, cỏ, đường ống, chim…, dựa theo sự thay đổi của 3 textureRegion là 3 con chim ta sẽ tạo được animation chim bay.



1. Trong project android, trong thư mục assets ta tạo mới 1 thư mục tên “data”, trong thư mục data ta chép hình texture.png và bỏ vào đó như hình dưới



1. Trong class AssetLoader khai báo các biến toàn cục và import các thư viện như sau:

**import** com.badlogic.gdx.graphics.Texture;

**import** com.badlogic.gdx.graphics.g2d.Animation;

**import** com.badlogic.gdx.graphics.g2d.TextureRegion;

**publicclass** AssetLoader {

**publicstatic** Texture *texture*;

**publicstatic** TextureRegion *bg*, *grass*;

**publicstatic** Animation *birdAnimation*;

**publicstatic** TextureRegion *bird*, *birdDown*, *birdUp*;

**publicstatic** TextureRegion *skullUp*, *skullDown*, *bar*;

}

1. Tiếp theo xây dựng hàm load và dispose như sau:

**publicstaticvoid** load() {

*texture* = **new** Texture(Gdx.*files*.internal("data/texture.png"));

*texture*.setFilter(TextureFilter.*Nearest*, TextureFilter.*Nearest*);

*bg* = **new** TextureRegion(*texture*, 0, 0, 136, 43);

*bg*.flip(**false**, **true**);

*grass* = **new** TextureRegion(*texture*, 0, 43, 143, 11);

*grass*.flip(**false**, **true**);

*birdDown* = **new** TextureRegion(*texture*, 136, 0, 17, 12);

*birdDown*.flip(**false**, **true**);

*bird* = **new** TextureRegion(*texture*, 153, 0, 17, 12);

*bird*.flip(**false**, **true**);

*birdUp* = **new** TextureRegion(*texture*, 170, 0, 17, 12);

*birdUp*.flip(**false**, **true**);

TextureRegion[] birds = { *birdDown*, *bird*, *birdUp* };

*birdAnimation* = **new** Animation(0.06f, birds);

*birdAnimation*.setPlayMode(Animation.PlayMode.*LOOP\_PINGPONG*);

*skullUp* = **new** TextureRegion(*texture*, 192, 0, 24, 14);

// Create by flipping existing skullUp

*skullDown* = **new** TextureRegion(*skullUp*);

*skullDown*.flip(**false**, **true**);

*bar* = **new** TextureRegion(*texture*, 136, 16, 22, 3);

*bar*.flip(**false**, **true**);

}

**publicstaticvoid** dispose() {

// We must dispose of the texture when we are finished.

*texture*.dispose();

}

Hàm load sẽ được gọi khi game bắt đầu chạy và hàm dispose sẽ được gọi khi game được đóng.

Trong hàm load đầu tiên đưa ảnh lớn vào texture. Sau đó dùng filter với TextureFilter.*Nearest* để khi phóng to hoặc thu nhỏ hình ảnh thì hình ảnh sẽ được xử lý để các điểm ảnh sẽ giữ lại hình dạng của nó thay gì bì mờ đi.

Sau đó ta dùng textureregion để cắt ra từng hình chữ nhật. Nó cần 5 đối số, cắt ra từ texture nào, tọa độ góc trên bên trái của texture region so với texture, và cuối cùng là kích thước rộng, cao của hình chữ nhật texture region.

Animation: tạo ra một mảng hình 3 con chim (xuống, bt, lên). Tạo thành 1 animation cho nó chạy liên tục và thay đổi theo 0.06 giây cuối cùng gán cách nó lặp theo kiểu Pingpong là qua lại qua lại liên tục (xuong, bt, len, bt, xuong, bt, len….)

# Gọi assets load

1. Mở lại class MyGdxGame trong project core trong package com.chao.conchimdien lên và trong hàm onCreate ta gọi load của AssetLoader, override hàm dispose và gọi dispose của assetloader.

**publicclass** MyGdxGame **extends** Game

{

@Override

**publicvoid** create() {

// **TODO** Auto-generated method stub

System.*out*.println("Game duoc tao ra ne");

AssetLoader.*load*();

setScreen(**new** GameScreen());

}

@Override

**publicvoid** dispose() {

// **TODO** Auto-generated method stub

**super**.dispose();

AssetLoader.*dispose*();

}

}

# Render hình ảnh lên GameRenderer

1. Mở file GameRenderer.java. Ta sẽ tạo ra SpriteBatch (nó giống ShapeRenderer) nó rút ra các hình ảnh cho chúng ta dùng và có các đối số là x,y, rong,cao, và loại. Ta còn cần phải thay đổi camera 136 và chiều cao của game sẽ do GameScreen quyết định do đó trong hàm tạo ta cũng cần nhận vào biến chiều cao. Theo mã bên dưới, chú ý các mã thêm hoặc thay đổi được đánh dấu với kí hiệu //\*

**publicclass** GameRenderer {

**private** GameWorld myWorld;

**private** OrthographicCamera cam;

**private** ShapeRenderer shapeRenderer;

**private** SpriteBatch batcher; //\*

**privateint**midPointY; //\*

**privateint**gameHeight; //\*

**public** GameRenderer(GameWorld world, **int** gameHeight, **int** midPointY) { //\*

// **TODO** Auto-generated constructor stub

myWorld = world;

cam = **new** OrthographicCamera();

**this**.gameHeight = gameHeight; //\*

**this**.midPointY = midPointY; //\*

cam.setToOrtho(**true**, 136, gameHeight); //\*

batcher = **new** SpriteBatch(); //\*

batcher.setProjectionMatrix(cam.combined); //\*

shapeRenderer = **new** ShapeRenderer();

shapeRenderer.setProjectionMatrix(cam.combined);

}

**publicvoid** render() {

System.*out*.println("GameRenderer - render");

Gdx.*gl*.glClearColor(0, 0, 0, 1);

Gdx.*gl*.glClear(GL20.*GL\_COLOR\_BUFFER\_BIT*);

}

}

# Sửa lớp GameScreen

Lưu lại thì GameScreen bị lỗi do truyền đối số sai, mở GameScreen lên ta phải truyền thêm chiều cao và midpoint và game, ngoài ra ta phải tạo thêm một biến runTime để lưu lại xem game đã chạy được bao lâu.

1. Mở File GameScreen lên, khai báo 1 biến toàn cục

**privatefloat**runTime; //\*

1. Trong hàm tạo GameScreen sửa phần renderer = **new** GameRenderer(world)thành:

renderer = **new** GameRenderer(world, (**int**) gameHeight, midPointY);

1. Trong hàm render ta thêm việc tăng thời gian bằng cách cộng dồn vào biến runTime sau đó ta truyền biến runtime vào cho renderer như sau:

@Override

**publicvoid** render(**float** delta) {

runTime += delta;

world.update(delta);

renderer.render(runTime);

}

# Sửa lại hàm render của GameRenderer để vẽ các đối tượng ra

1. Mở lại file GameRender, trong hàm render sửa lại để nhận vào đối số runTime và thêm như bên dưới.

**publicvoid** render(**float** runTime) {

System.*out*.println("GameRenderer - render");

Gdx.*gl*.glClearColor(0, 0, 0, 1);

Gdx.*gl*.glClear(GL20.*GL\_COLOR\_BUFFER\_BIT*);

// Tasedichuyenconchimrangoaidetoiuuhoahieuxuatsau

Bird bird = myWorld.getBird();

// bat dau ShapeRenderer

shapeRenderer.begin(ShapeType.*Filled*);

// ve Background color

shapeRenderer.setColor(55 / 255.0f, 80 / 255.0f, 100 / 255.0f, 1);

shapeRenderer.rect(0, 0, 136, midPointY + 66);

// ve Grass (co)

shapeRenderer.setColor(111 / 255.0f, 186 / 255.0f, 45 / 255.0f, 1);

shapeRenderer.rect(0, midPointY + 66, 136, 11);

// ve Dirt (ve dam nhacuava may)

shapeRenderer.setColor(147 / 255.0f, 80 / 255.0f, 27 / 255.0f, 1);

shapeRenderer.rect(0, midPointY + 77, 136, 52);

// ketthuc ShapeRenderer

shapeRenderer.end();

// bat dau SpriteBatch

batcher.begin();

// vohieuhoa transparency (tinhtrongsuot)

// dieu nay can thietchoviectoiuutoc do khikhong can phaive

//cachinhyeucau transparency

batcher.disableBlending();

batcher.draw(AssetLoader.*bg*, 0, midPointY + 23, 136, 43);

// conchimthi can transparency

batcher.enableBlending();

// vechimtaitheotoa do. truyxuattu AssetLoader

// truyenbien runTime de lay frame hientai

batcher.draw(AssetLoader.*birdAnimation*.getKeyFrame(runTime),

bird.getX(), bird.getY(), bird.getWidth(), bird.getHeight());

// ketthuc SpriteBatch

batcher.end();

}

1. Mở project conchimdien-desktop, click phải file DesktopLauncher.java chọn run as->java application để chạy. Thấy rằng con chim sẽ rớt từ trên xuống, click vào để thấy nó nhảy tưng tưng giống 1 con điên.

