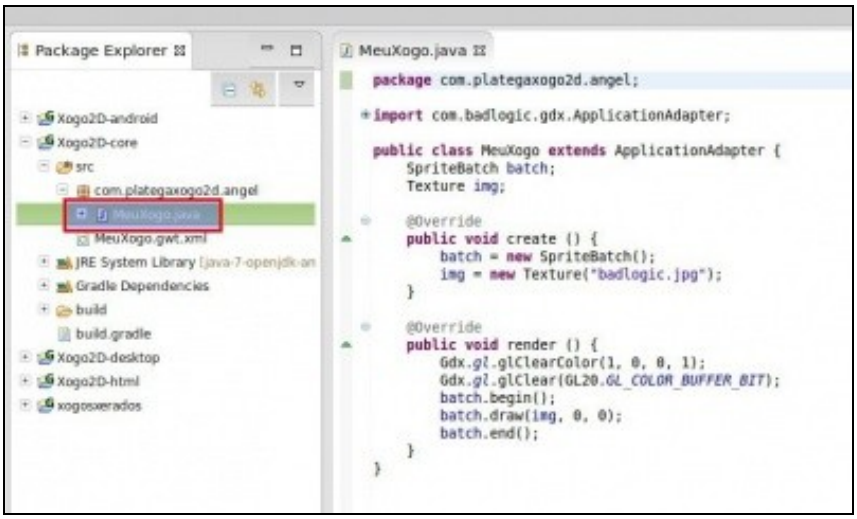
LIBGDX Analizando o proxecto común

UNIDADE 2: Analizando o proxecto común

Ciclo de vida do noso xogo

Como xa comentamos neste entrada, o framework vai xerar un proxecto para cada plataforma e un proxecto 'común' o que van facer referencia todos eles.

Este proxecto común é o **Xogo2D-Core**. É a clase que van facer uso todos os proxectos vai ser **MeuXogo.java** que é o nome que indicamos cando xeramos os proxectos coa ferramenta gráfica.



Veremos máis adiante que imos poder chamar a clases que deriven de **ApplicationAdapter** como no exemplo ou ben que deriven da clase **Game**.

Imos analizar o código moi por enriba da clase xerada:

```
package com.plategaxogo2d.angel;

import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;

public class MeuXogo extends ApplicationAdapter {
SpriteBatch batch;
Texture img;

@Override
public void create () {
batch = new SpriteBatch();
img = new Texture("badlogic.jpg");
}

@Override
public void render () {
Gdx.gl.glClearColor(1, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
batch.begin();
batch.draw(img, 0, 0);
batch.end();
}
}
```

- O primeiro (liña 9) é crear unha clase que derive da clase ApplicationAdapter. Ó facelo podemos sobrescribir os seguintes métodos:

```
@Override
public void create() {
                    }               // Chamado cando se crea a instancia.
@Override
public void dispose() {
                     }              // Chamado cando se libera. Antes sempre se chama a pause.
@Override
public void render() {
                                    // Chamado continuamente para facer o render do xogo.
                    }               // Aquí tamén iría a lóxica do programa
@Override
public void resize(int width, int height) {
                                    // Chamado se cambia a resolución da pantalla (tamén se chama unha vez despois do create)
                    }
@Override
public void pause() {
                                    // Chamado ó saír da aplicación.
                   }                //  En Android cando se preme o boton Home ou ten unha chamada. Bo sitio para gardar o estado d
@Override
public void resume() {
                    }              // Chamado ó reanudar ó xogo (volvemos ó xogo en Android ou vimos de minimizar o xogo en PC)
```

Nota: O que vai facer o framework é chamar continuamente ó método render e o que faremos nese método será borrar toda a pantalla e voltar a debuxar todos os elementos do noso xogo.

Imos comprobar como se van chamar ós diferentes métodos durante o ciclo de vida do noso xogo. Para velo imos usar un método que amosa na **ventá de log** do Eclipse os datos que lle pasemos.

Creamos dentro do paquete onde se atopa a clase MeuXogo (com.plategaxogo2d.o_voso_nome) unha clase de nome Utiles.

Dentro de dita clase definimos unha constante de clase cun valor que vai ser a etiqueta coa que identificaremos as mensaxes enviadas por nos ó Eclipse.

```
private static final String LOG = "XOGO2D";
```

Despois definimos un método de clase o que se lle vai pasar o nome da clase dende onde se chama, o nome do método e a mensaxe a imprimir.

```
/**
         * Método para imprimir mensaxes de log no Eclipse.
         * Usado para depuración (debugger)
         * @param clase: nome da clase de onde se chama
         * @param metodo: nome do método de onde se chama
         * @param mensaxe: mensaxe a imprimir
         */
public static void imprimirLog(String clase, String metodo, String mensaxe){
Gdx.app.log(LOG, clase + ":"+metodo+":"+mensaxe);
}
```

Agora xa podemos sobrescribir os diferentes métodos e facer o log de cada un deles. O faremos co método **create** que trae a clase.

Aviso: Non imprimir ningún log no método **render** xa que dito método é chamado de forma continua (de 25-80 veces por segundo) polo framework.

O código completo quedaría así:

Codigo da clase Utiles:

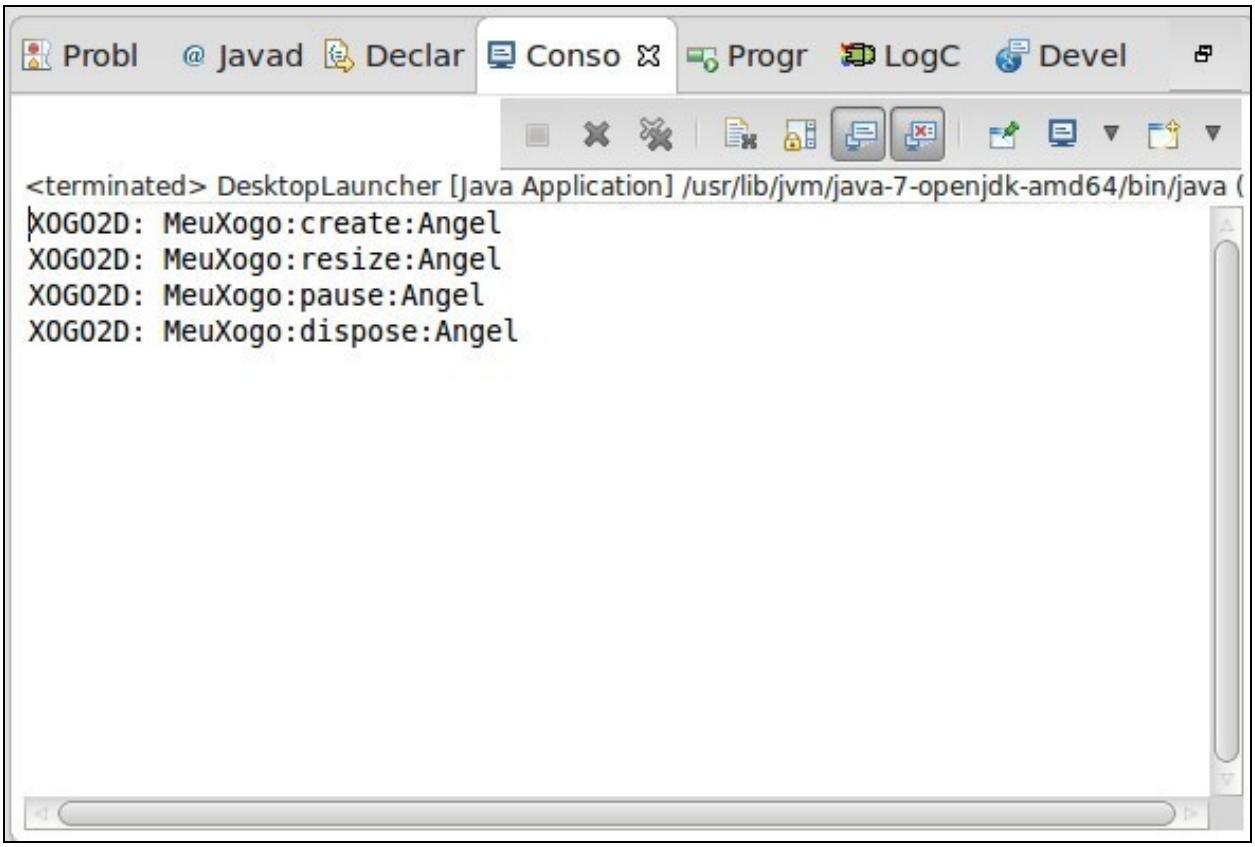
```
package com.plategaxogo2d.angel;

import com.badlogic.gdx.Gdx;

public class Utiles {
private static final String LOG = "XOGO2D";

/**
     * Método para imprimir mensaxes de log no Eclipse.
```

```
     * Usado para depuración (debugger)
     * @param clase: nome da clase de onde se chama
     * @param metodo: nome do método de onde se chama
     * @param mensaxe: mensaxe a imprimir
     */
    public static void imprimirLog(String clase, String metodo, String mensaxe){
            Gdx.app.log(LOG, clase + ":"+metodo+":"+mensaxe);
    }
}
```

Codigo da clase MeuXogo:

```
package com.plategaxogo2d.angel;

import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;

public class MeuXogo extends ApplicationAdapter {
SpriteBatch batch;
Texture img;

@Override
public void create () {
batch = new SpriteBatch();
img = new Texture("badlogic.jpg");

Utiles.imprimirLog("MeuXogo", "create", "Angel");
}

@Override
public void render () {
Gdx.gl.glClearColor(1, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
batch.begin();
batch.draw(img, 0, 0);
batch.end();

}

}
```

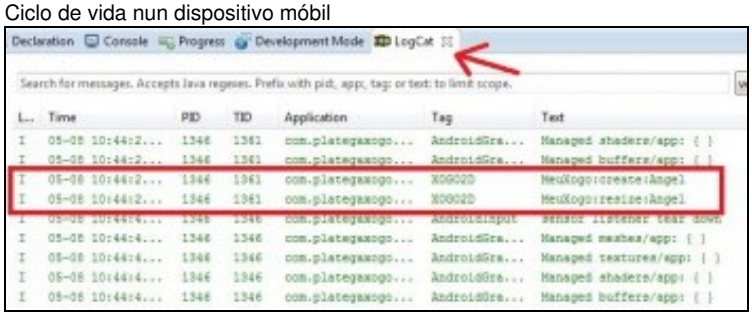
Se executamos a versión Desktop e pechamos o xogo podemos comprobar como aparecen na ventá de consola as mensaxes postas (neste caso xa están sobrescritos os métodos indicados anteriormente menos o render):

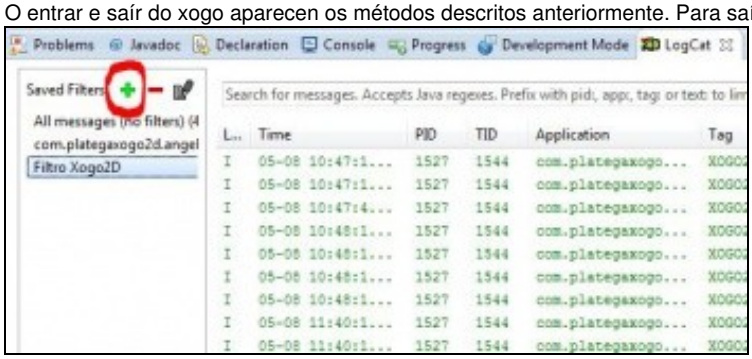


Nota: A ventá Console se pode amosar indo o menú de Eclipse Window => Show View => Console.

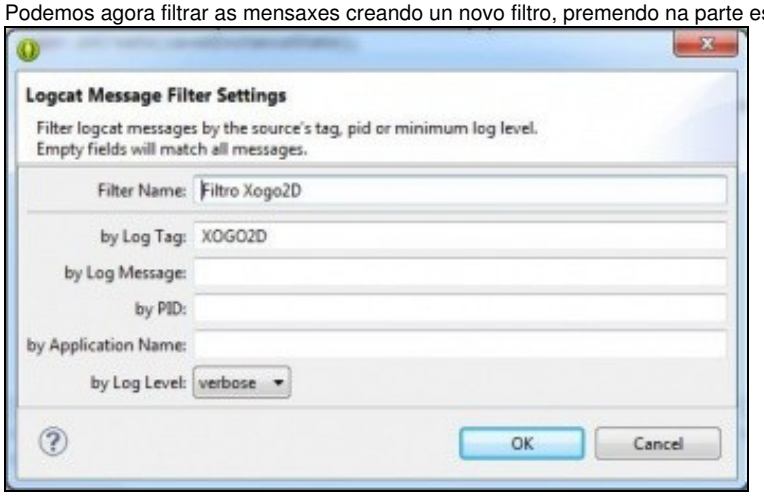
Se o facemos na versión Android as mensaxes non aparece na ventá Console senón na ventá de *LogCat*.

Nota: Dita ventá se atopa no menú de Eclipse Window => Show View => Other e escoller a opción *LogCat* como se va na seguinte imaxe:

- Ciclo de vida nun dispositivo móbil
- 

O entrar e saír do xogo aparecen os métodos descritos anteriormente. Para saír do xogo debemos de premer a tecla *Back* do dispositivo.
- 

Podemos agora filtrar as mensaxes creando un novo filtro, premendo na parte esquerda no símbolo + (cor verde).

- 

Os datos se filtran en base o Tag que se corresponde có valor da propiedade LOG.

- 

Vemos agora só os eventos do noso xogo.

- 

Se en vez de premer a tecla Back prememos a tecla *Casa* no emulador ou cambiamos de aplicación nun dispositivo real.

- 

Podemos observar como non executa o método dispose. Ó voltar a executar o xogo pasa polo método resume.

TAREFA 2.0 A FACER: Esta parte está asociada á realización dunha tarefa.

Utilidade FPS

Outra utilidade que imos ter para desenvolver os nosos xogos é comprobar a que velocidade (fotogramas por segundo = fps) ó que se executa o noso xogo.

Definimos a propiedade fps na clase Xogo2D-core:

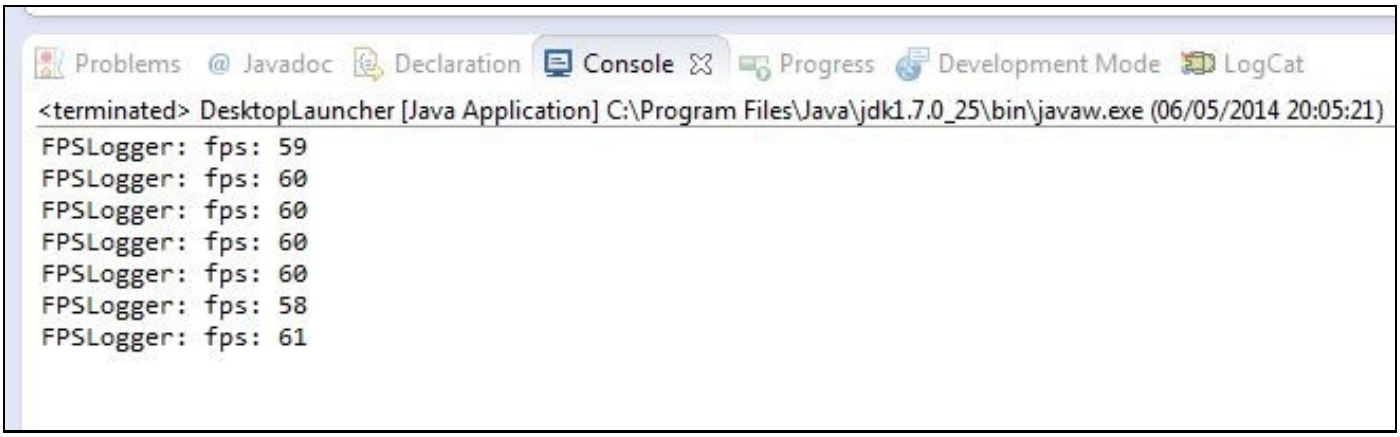
```
private FPSLogger fps;
```

Nota: Ó poñer dita liña dará un erro xa que a clase FPSLogger non está importada. Premer a combinación de teclas Ctrol+Shift+O para importalo automaticamente.

Modificamos os métodos **create** e **render**:

```
@Override
public void create() {
  fps = new FPSLogger();
.............
@Override
public void render() {
  fps.log();
..............
```

Podemos ver na lapela de **Console** se estamos a executar a versión Desktop os fotogramas por segundo.



A velocidade para ter un xogo 'xogable' vai depender do tipo de xogo, aínda que con 25fps debería chegar, pero dependerá do tipo de xogo e da persoa.

Podemos consultar nesta dirección http://www.tweakguides.com/Graphics_5.html máis información sobre os fps.