libGDX

https://github.com/libgdx/libgdx/wiki/

Rafael Vieira Coelho

rafael.coelho@farroupilha.ifrs.edu.br

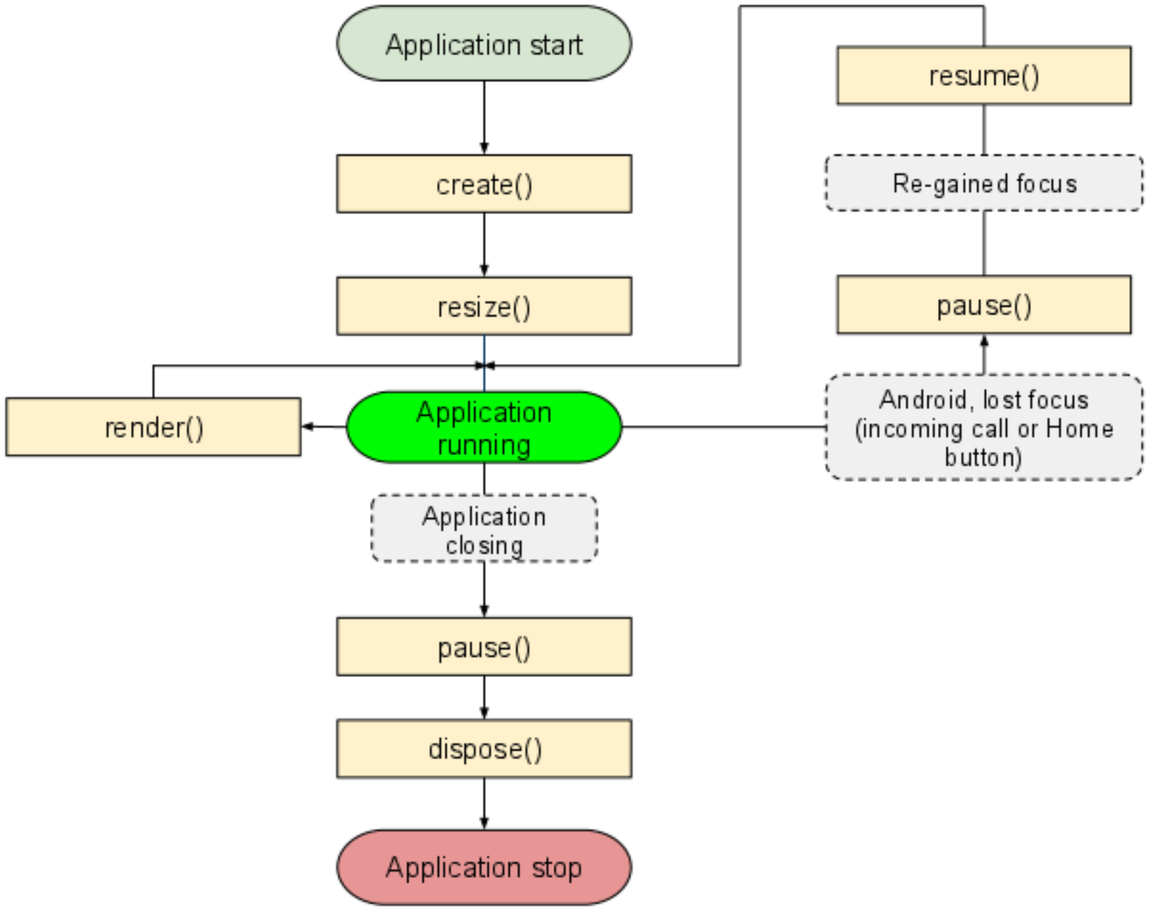
Tópicos

- **The Application Framework**
- A Simple Game
- File Handling
- Networking
- Preferences
- Input Handling
- Memory Management
- Audio
- Graphics

https://github.com/libgdx/libgdx/wiki/

O Ciclo de Vida da Aplicação

- Primeiramente, deve-se implementar a interface **ApplicationListener**
- Sempre que ocorrer um evento ocorrer, será chamado um dos métodos ao lado. O método **render** é chamado em *loop* até fechar a aplicação.




```
public class MyGame implements ApplicationListener {
    public void create () {
    }

    public void render () {
    }

    public void resize (int width, int height) {
    }

    public void pause () {
    }

    public void resume () {
    }

    public void dispose () {
    }
}
```

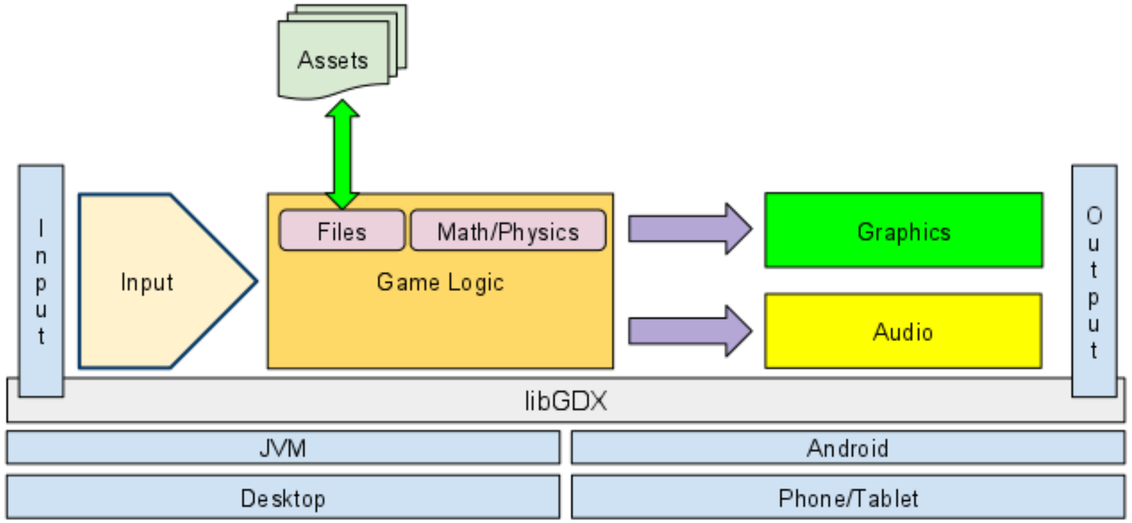
Programa Principal

- Teremos uma classe que tem o programa principal e sua única tarefa é instanciar um objeto da classe **LwjglApplication** a partir de uma configuração e da classe da implementação do jogo (no exemplo MyGame)

```
public class DesktopStarter {
   public static void main(String[] argv) {
      LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
      new LwjglApplication(new MyGame(), config);
   }
}
```

Módulos

- LibGDX é composta por diversos módulos que permitem interação com o SO:
- **Application**: roda a aplicação e informa a API cliente sobre eventos como mudança de tamanho da janela, uso de memória, log, etc.
- **Files**: cria uma camada de abstração ao sistema de arquivos do SO.
- **Input**: informa a API cliente sobre entrada de dados (teclado, mouse ou touch).
- **Net**: provê meios de ter acesso a recursos como HTTP/HTTPS e criar servidores e clientes socket via TCP.
- **Audio**: permite tocar efeitos sonoros e músicas.
- **Graphics**: permite o uso de OpenGL 2.0 quando disponível.




```
public class DesktopStarter {
   public static void main(String[] argv) {
      LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
      new LwjglApplication(new MyGame(), config);
   }
}
```

Acessando Módulos

- Para acessar os módulos descritos previamente, acessar campos estáticos da classe **Gdx**.
- Por exemplo, para acessar o módulo **audio**:

```
// creates a new AudioDevice to which 16-bit PCM samples can be written
AudioDevice audioDevice = Gdx.audio.newAudioDevice(44100, false);
```

- Para acessar outros módulos, é feito de maneira similar: **Gdx.app** para acessar o módulo Application, **Gdx.files** para acessar o módulo Files, etc.

Módulo Input

- No caso de uma aplicação desktop, o touchscreen é substituído pelo mouse.
- Permite obter o estado das teclas e registrar processadores de entrada para usar um modelo de entrada baseado em eventos.
- Ex: o código abaixo obtém as coordenadas atuais no momento de um click do mouse:

```
if (Gdx.input.isTouched()) {
  System.out.println("Input occurred at x=" + Gdx.input.getX() + ", y=" + Gdx.input.getY());
}
```

Módulo Graphics

- Este módulo provê a comunicação com a GPU e tem métodos para obter instancias dos wrappers de OpenGL.

- Também permite gerar **Pixmaps** e **Textures**.

- O método **getGL20** retorna um objeto da classe GL20 e representa o OpenGL API 2.0.

```
GL20 gl = Gdx.graphics.getGL20 ();
```

- A partir deste objeto, podemos desenhar na tela.

- O exemplo abaixo limpa a tela e a pinta de vermelho.

```
gl.glClearColor(1f, 0.0f, 0.0f, 1);
gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
```

- LibGDX suporta as versões:

GL Version	*Method to access*
2.0	`Gdx.graphics.getGL20();`
3.0	`Gdx.graphics.getGL30();`

Módulo Files

- Permite acessar (ler/escrever) arquivos, independente da plataforma.
- Um exemplo de uso é para carregar imagens de **assets** (texturas, sons).
- Também pode ser usado para salvar o **estado do jogo** ou os ***high scores***.
- Os arquivos são armazenados na pasta assets do projeto.

```
Texture myTexture = new Texture(Gdx.files.internal("assets/textures/brick.png"));
```

Módulo Audio

- Suporta arquivos com formato WAV, MP3 e OGG.

- Para tocar o arquivo myMusicFile.mp3 com volume pela metade em loop, basta:

```
Music music = Gdx.audio.newMusic(Gdx.files.getFileHandle("data/myMusicFile.mp3",("data/myMusicFile.mp3", FileType.Internal));
music.setVolume(0.5f);
music.play();
music.setLooping(true);
```

Módulo Networking

- É usado para jogos em rede e pode ser usado para adicionar jogadores (multiplayer), enviar jogadores para um site, etc.
- Para criar requisições GTTP, cria-se um objeto **HTTpRequestBuilder:**

```
HttpRequestBuilder requestBuilder = new HttpRequestBuilder();
HttpRequest httpRequest = requestBuilder.newRequest()
    .method(HttpMethods.GET)
    .url("http://www.google.de")
    .build();
Gdx.net.sendHttpRequest(httpRequest, httpResponseListener);
```

- Já para criar requisições HTTP com argumentos:

```
HttpRequestBuilder requestBuilder = new HttpRequestBuilder();
HttpRequest httpRequest = requestBuilder.newRequest()
    .method(HttpMethods.GET)
    .url("http://www.google.de")
    .content("q=libgdx&example=example")
    .build();
Gdx.net.sendHttpRequest(httpRequest, httpResponseListener);
```

Configurações Iniciais

Desktop (LWJGL)

Opening the `Main.java` class in `my-gdx-game` shows the following:

```
package com.me.mygdxgame;

import com.badlogic.gdx.backends.lwjgl.LwjglApplication;
import com.badlogic.gdx.backends.lwjgl.LwjglApplicationConfiguration;

public class Main {
   public static void main(String[] args) {
      LwjglApplicationConfiguration cfg = new LwjglApplicationConfiguration();
      cfg.title = "my-gdx-game";
      cfg.useGL30 = false;
      cfg.width = 480;
      cfg.height = 320;

      new LwjglApplication(new MyGdxGame(), cfg);
   }
}
```

Desktop (LWJGL3)

LWJGL3 is the modern backend recommended for new projects. Unfortunately, it is not set as the default in the setup project. To enable it, you must change the following line in the build.gradle for your project:

From:

```
api "com.badlogicgames.gdx:gdx-backend-lwjgl:$gdxVersion"
```

To:

```
api "com.badlogicgames.gdx:gdx-backend-lwjgl3:$gdxVersion"
```

Make sure to refresh your Gradle dependencies in your IDE. You will have to make the following changes to your DesktopLauncher class as well:

```
package com.me.mygdxgame;

import com.badlogic.gdx.backends.lwjgl3.Lwjgl3Application;
import com.badlogic.gdx.backends.lwjgl3.Lwjgl3ApplicationConfiguration;

public class Main {
   public static void main(String[] args) {
      Lwjgl3ApplicationConfiguration config = new Lwjgl3ApplicationConfiguration();
      config.setTitle("my-gdx-game");
      config.setWindowedMode(480, 320);

      new Lwjgl3Application(new MyGdxGame(), config);
   }
}
```

Plataformas Disponíveis

- Podemos implementar o nosso jogo para Android, Desktop e Web.

- Para descobrir o tipo de aplicação que está rodando, basta chamar o método **getType** do módulo **app**.

```
switch (Gdx.app.getType()) {
    case Android:
        // android specific code
        break;
    case Desktop:
        // desktop specific code
        break;
    case WebGl:
        // HTML5 specific code
        break;
    default:
        // Other platforms specific code
}
```

- Desta forma, podemos fazer adaptações quando necessário.
- No Android, é possível descobrir também a versão:

```
int androidVersion = Gdx.app.getVersion();
```

Consumo de Memória

- Para fazer testes e depuração, pode ser útil ter acesso a dados do consumo de memória tanto do heap de Java quando do nativo.

```
long javaHeap = Gdx.app.getJavaHeap();
long nativeHeap = Gdx.app.getNativeHeap();
```

- Eles retornam o número de bytes em uso no momento.

Logging

- O módulo Application permite criar arquivos de logs.

```
Gdx.app.log("MyTag", "my informative message");
Gdx.app.error("MyTag", "my error message", exception);
Gdx.app.debug("MyTag", "my debug message");
```

- Dependendo da plataforma, as mensagens de log são colocadas no console no caso de uma aplicação Desktop.

- E o nível de log pode ser especificado:

```
Gdx.app.setLogLevel(logLevel);
```

where `logLevel` can be one of the following values:

- Application.LOG_NONE: mutes all logging.
- Application.LOG_DEBUG: logs all messages.
- Application.LOG_ERROR: logs only error messages.
- Application.LOG_INFO: logs error and normal messages.

Threading

- Todos métodos da interface ApplicationListener são chamados na mesma Thread.
- Para passar dados de uma Thread para a Thread de renderização (principal), deve-se usar o método **Application.postRunnable**.
- Para múltiplas threads usar componentes gráficos, deve-se usar componentes **scene2D**.

```
new Thread(new Runnable() {
   @Override
   public void run() {
      // do something important here, asynchronously to the re
      final Result result = createResult();
      // post a Runnable to the rendering thread that processe
      Gdx.app.postRunnable(new Runnable() {
         @Override
         public void run() {
            // process the result, e.g. add it to an Array<Res
            results.add(result);
         }
      });
   }
}).start();
```