www.gameutil2d.org

[Versão Java 2.0]

Criado por: Luciano Alves da Silva

Guia de Referência

[Maio 2018]

Introdução: Sobre o Framework GameUtil2D

O framework **GameUtil2D** é um conjunto de classes e métodos que permite a criação de jogos em 2D para a plataforma Java (também existindo uma versão para a linguagem Javascript).

Características do Framework

O framework foi desenvolvido seguindo uma "filosofia": Permitir a criação de jogos de uma maneira simples, fácil e direta, sem a necessidade de criar grandes códigos para obter resultados significativos (além de possuir uma sintaxe clara e compreensiva).

Ele oferece diversas classes e métodos prontos para a criação de um jogo. Entre elas podemos citar classes para a criação de sprites (estáticos e animados), detecção de colisões, criação de personagens, detecção de dispositivos de entrada e saída (mouse e teclado), criação de cenários e etc.

Como esse framework surgiu?

Em 2012, Luciano Alves da Silva (na época professor de uma escola de desenvolvimento de jogos do Rio de Janeiro), estudava sobre os mais diversos frameworks para a criação de jogos (na época para a plataforma Android) e percebeu que a maioria das ferramentas eram complexas (e com carência de tutoriais), o que dificultava o aprendizado. A primeira versão do GameUtil2D desenvolvida surgiu para a plataforma Android, com uma série e classes e métodos facilitando a construção de jogos, eliminando a complexidade do desenvolvimento. Hoje esse framework encontra-se disponível somente para Java (Desktop Windows, Linux, Mac) e Javascript (Web e Mobile). Futuramente será disponível também em C# para o Visual Studio.

Capítulo 1: Instalação e configuração do Framework

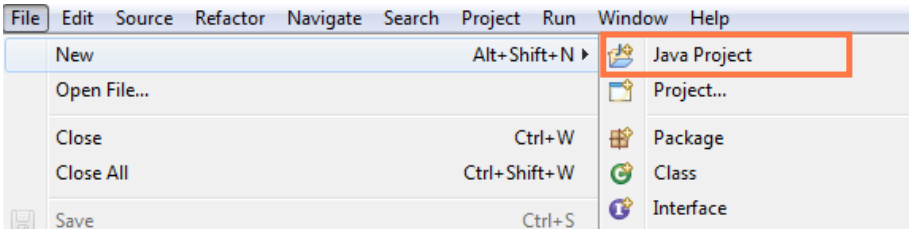
Essa documentação mostrará como instalar o GameUtil2D (Versão 2.0) para a plataforma Java. O framework consiste em um arquivo ".jar" (Java Archive File), com as classes necessárias para a construção de jogos. Serão mostradas aqui como instalar o framework nas ferramentas Eclipse e NetBeans (caso você trabalhe com outra ferramenta de desenvolvimento, o procedimento é similar).

Versão do Java

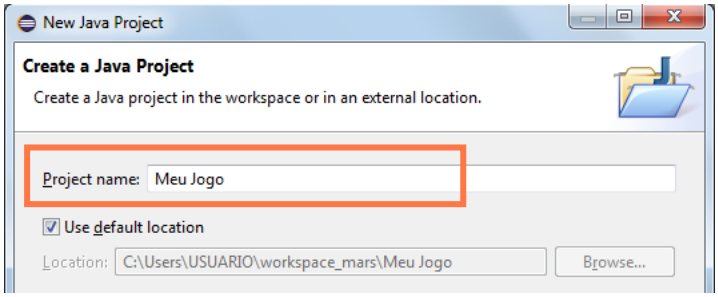
A versão mínima do Java a ser utilizada para o desenvolvimento com o framework é 1.8.0.111.

Instalando no Eclipse

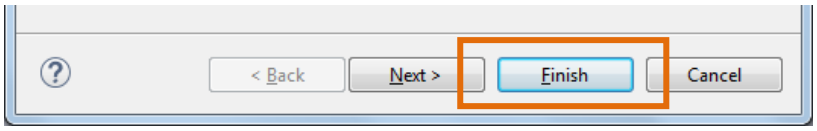
Crie um novo projeto no Eclipse (Java Project), usando o "File" / "New" / "Java Project".



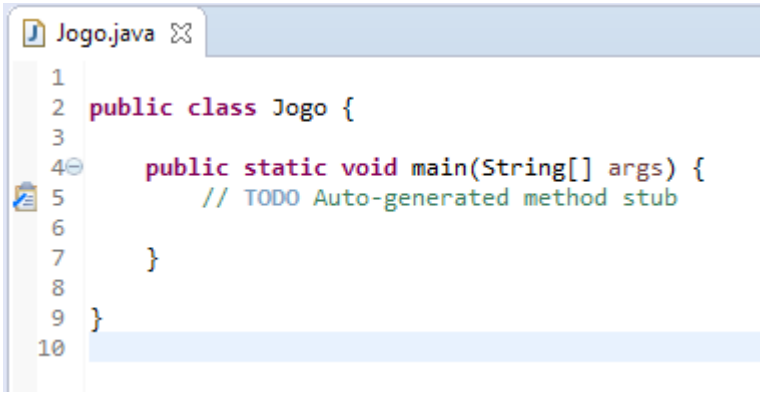
Em seguida, dê um nome para seu projeto:



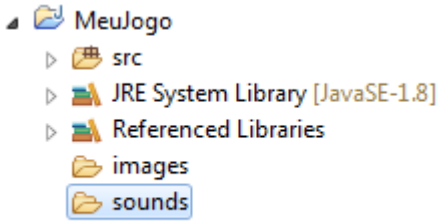
Em seguida, clique no botão “Finish” para criar o projeto



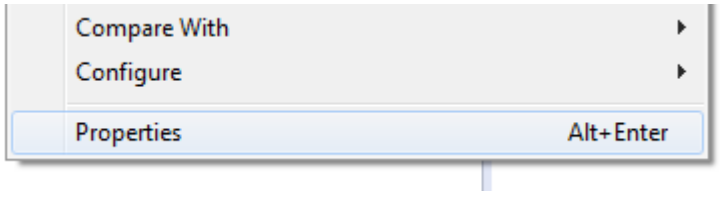
Feito isso, crie uma nova classe para o projeto (incluindo um método “main”). Vejamos um exemplo abaixo:



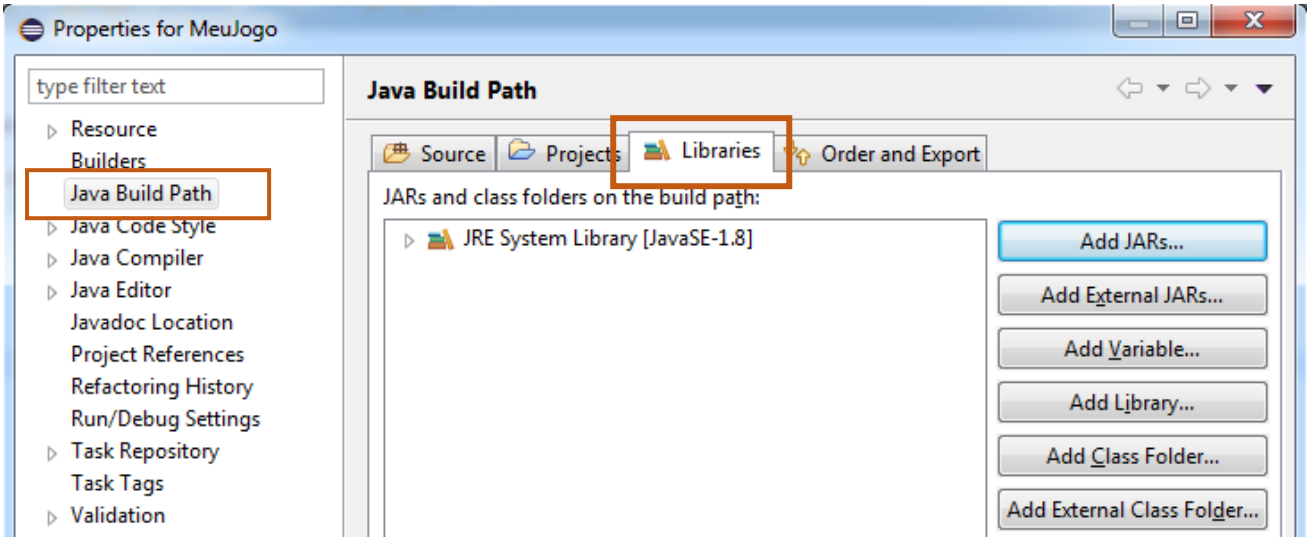
Crie dentro crie dentro do seu projeto dois diretórios: “images” (onde ficam armazenadas as imagens do jogo) e sounds (onde ficam armazenadas as músicas e sons de efeito).



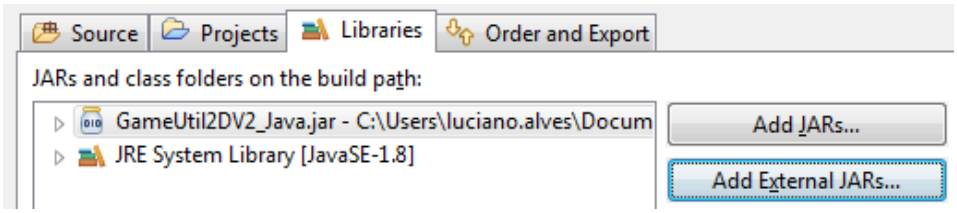
Vamos importar o arquivo do framework GameUtil2D (arquivo “.jar”) para dentro do seu projeto. Clique com o botão direito do mouse sobre o projeto criado e selecione “Properties”:



Selecione a opção “Java Build Path” e certifique-se de que a guia “Libraries” esteja selecionada.

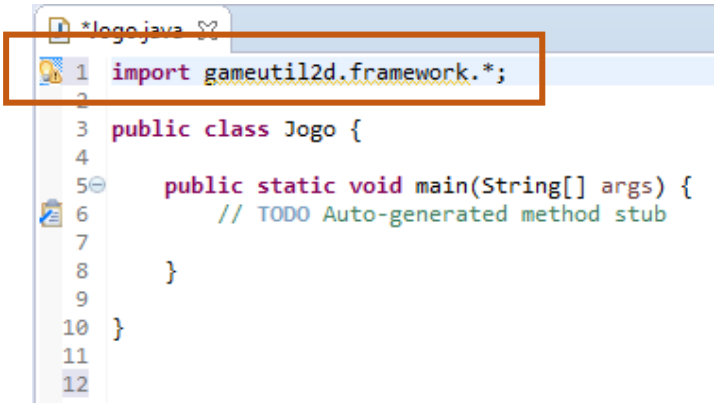


Clique no botão “Add External JARs...” e adicione o framework (arquivo “GameUtil2DVx.jar”) dentro do nosso projeto. Veja o resultado:



Clique em “OK” para confirmar

Volte para a classe que você criou e adicione o seguinte **import**:



Implemente na classe que criamos a interface **Game** com os seus respectivos métodos**:**

*Jogo.java

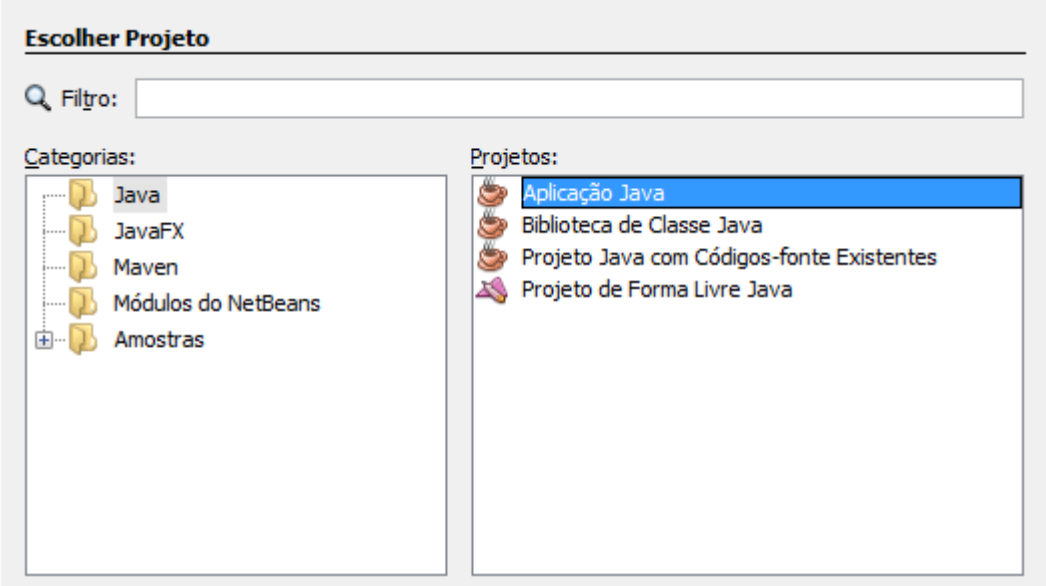
```
1  import java.awt.Graphics;
2
3  import gameutil2d.classes.input.KeyboardState;
4  import gameutil2d.classes.input.MouseState;
5  import gameutil2d.framework.*;
6
7  public class Jogo implements Game {
8
9      public static void main(String[] args) {
10
11
12     }
13
14     @Override
15     public void Initialize() {
16
17
18     }
19
20     @Override
21     public void Update(KeyboardState keyboardState, MouseState mouseState) {
22
23
24     }
25
26     @Override
27     public void Draw(Graphics graphics) {
28
29
30     }
31
32
33 }
```

Para que o jogo seja executado, coloque no método **main** o seguinte código:

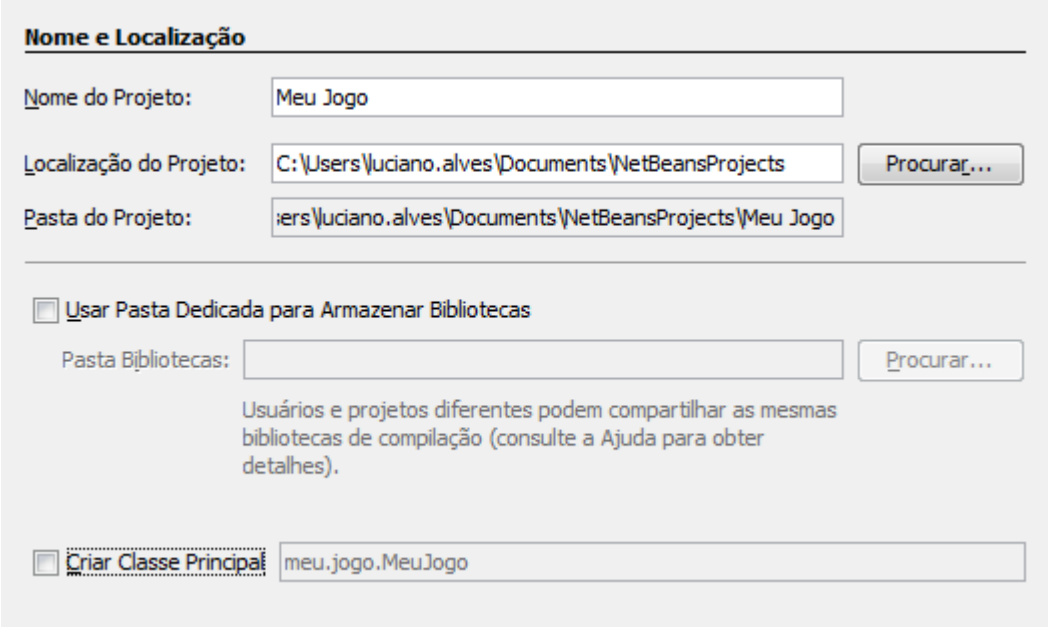
```
7  public class Jogo implements Game {
8
9      public static void main(String[] args) {
10
11         new GameUtil2D(new Jogo());
12
13     }
14
```

Instalando no NetBeans

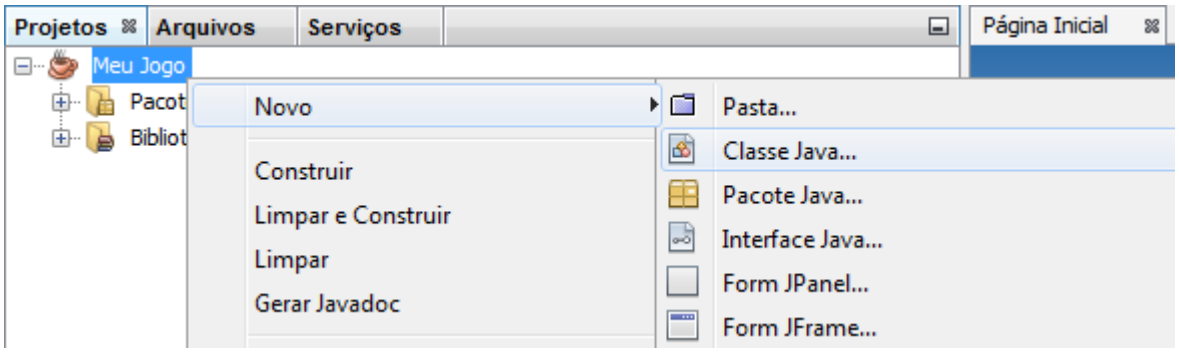
Com o NetBeans aberto, crie um novo projeto do tipo “Aplicação Java”:

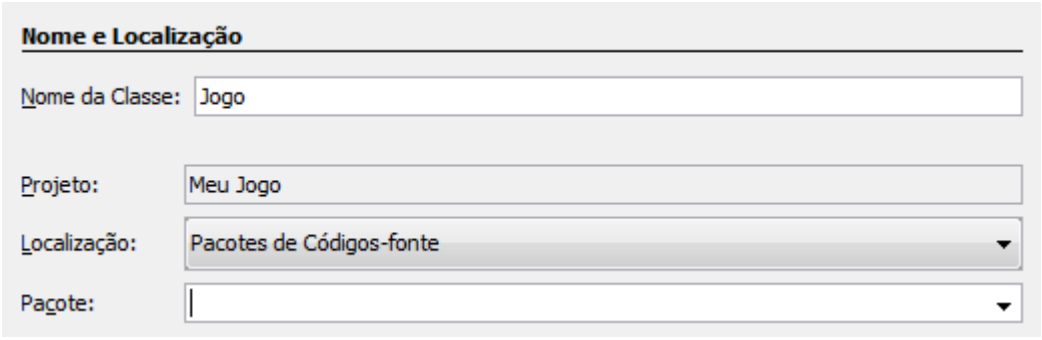


Dê um nome para o projeto que representará seu jogo, como por exemplo: “Meu Jogo”:

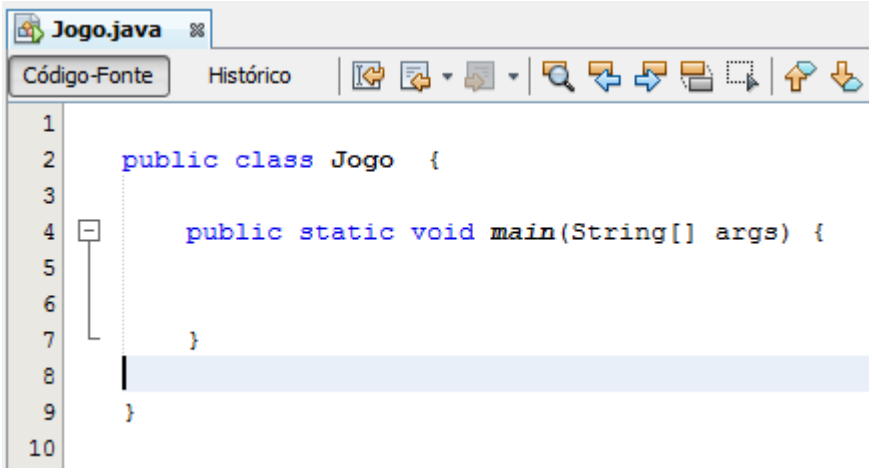


Agora vamos criar uma nova classe Java dentro do nosso projeto no NetBens:

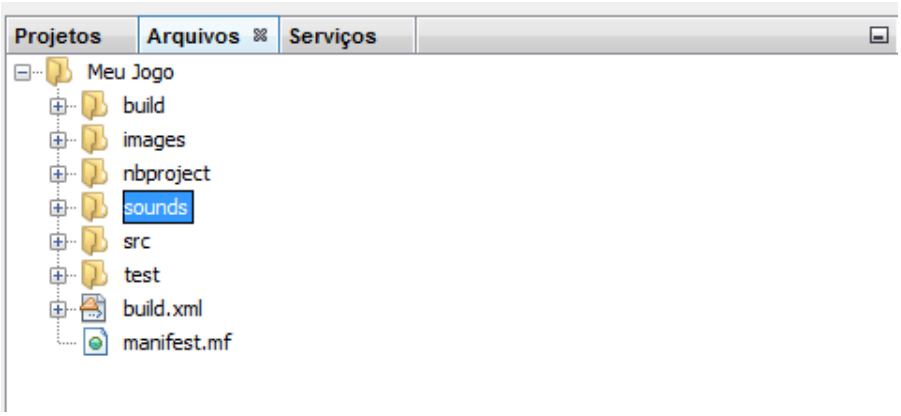




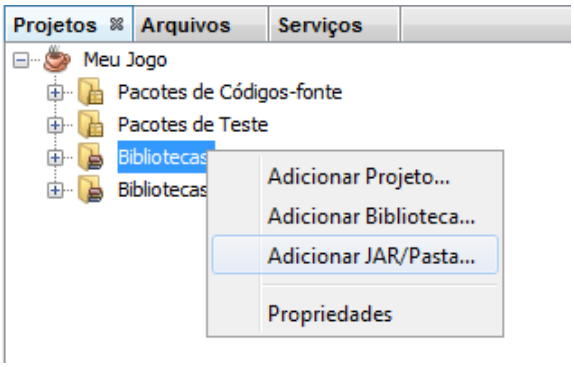
Crie a seguinte estrutura de código básica na classe adicionada ao seu projeto:



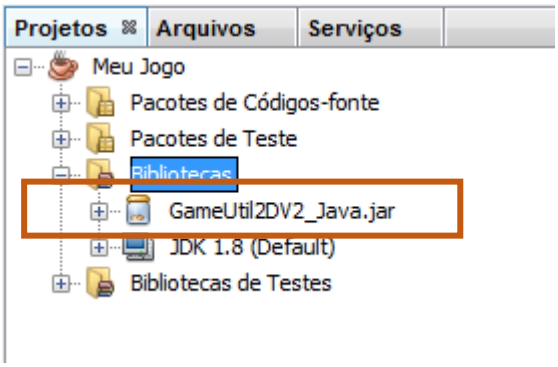
Crie dentro crie dentro do seu projeto dois diretórios: “images” (onde ficam armazenadas as imagens do jogo) e sounds (onde ficam armazenadas as músicas e sons de efeito).



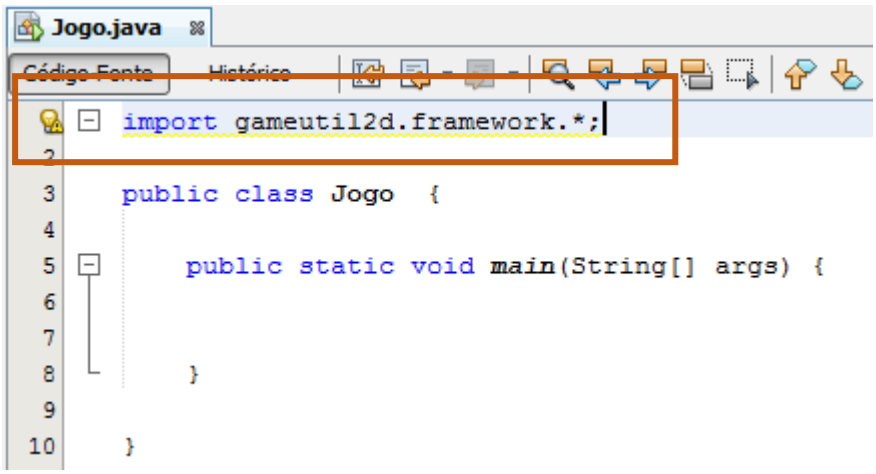
Agora vamos adicionar o arquivo “.jar” referente ao framework. Na seção “Projetos”, clique com o botão direito sobre a pasta “Bibliotecas” e selecione a opção “Adicionar Jar/Pasta”.



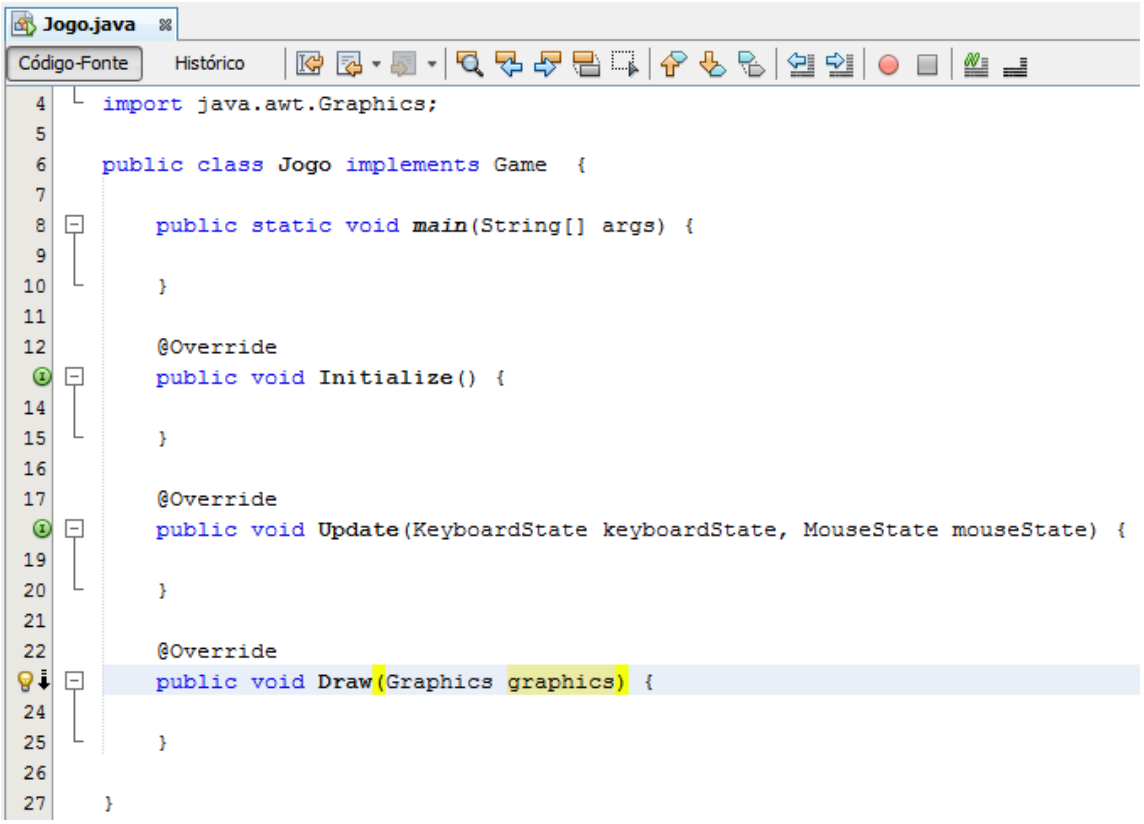
Selecione o arquivo referente ao framework e adicione ao projeto



No código da classe, vamos importar o seguinte pacote:



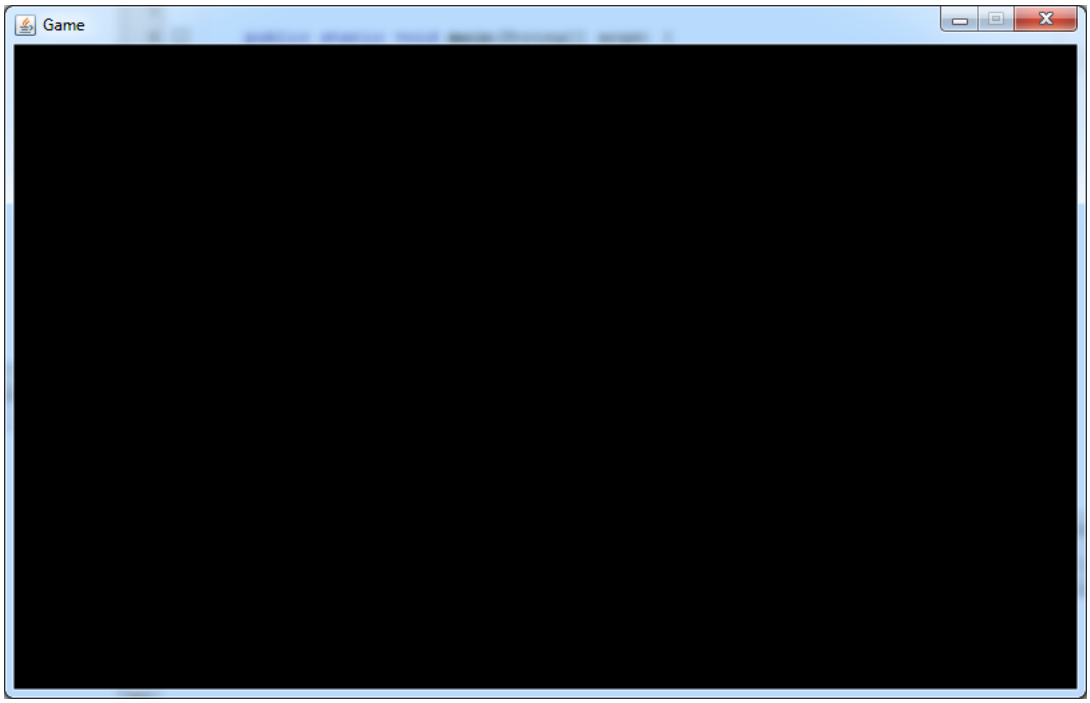
Implemente a interface **Game** em nossa classe com os seus respectivos métodos a seguir:



Para que o jogo seja executado, coloque no método **main** o seguinte código:

```
6   public class Jogo implements Game  {
7
8       public static void main(String[] args) {
9           new GameUtil2D(new Jogo());
10      }
```

Tanto na configuração das classes feita no Eclipse, NetBeans (ou em outra ferramenta de desenvolvimento Java), teremos o seguinte resultado ao executarmos o jogo:



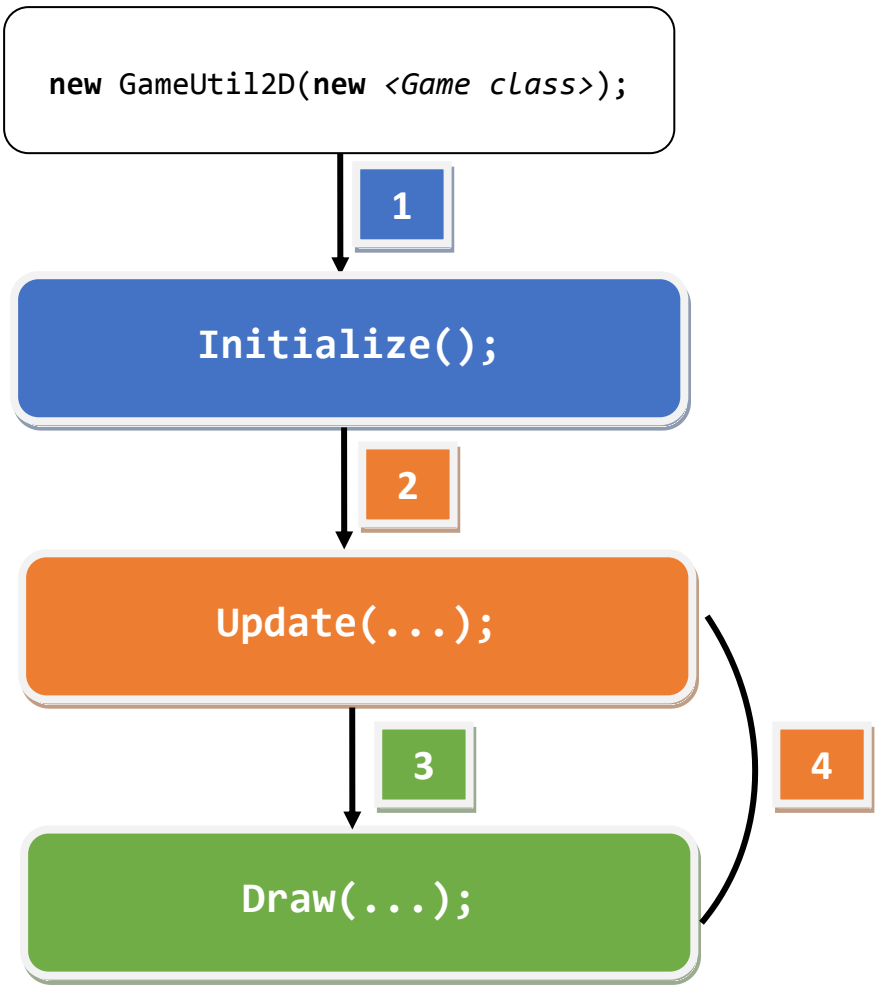
Descrição dos métodos adicionados

Vamos entender agora os métodos da interface **Game** (do GameUtil2D) que foram adicionados a sua classe.

Método	Descrição
`public void Initialize()`	Esse método é executado quando o jogo é carregado na memória. Nesse método iniciamos todos os objetos (como imagens, sons e etc.)
`public void Update(KeyboardState keyboardState,MouseState mouseState)`	Método responsável por executar a lógica do jogo (como movimentos, ações e etc.)
`public void Draw(Graphics graphics)`	Método responsável por desenhar todos os elementos do jogo na tela.

Ciclo de execução os métodos do Jogo

Vamos entender agora o ciclo de execução dos métodos do GameUtil2D adicionamos em nossa classe através do diagrama abaixo:



1) Após o carregamento da instância da classe GameUtil2D, é executado o método **Initialize,** responsável por carregar todos os objetos do jogo.

2) Depois de processado o método **Initialize**, será executado o método **Update** responsável por processar a lógica do jogo.

3) Depois de processado o método **Update**, será executado o método **Draw** responsável por desenhar na tela as atualizações das ações do jogo.

4) Depois de processado o método **Draw**, retornamos ao método **Update**, processamento novamente a lógica do jogo. E assim segue o ciclo **Draw** e **Update**.

Capítulo 2: Guia de referência – Classes e métodos

Nesse capítulo será mostrado um guia completo que mostra todas as classes de pacotes do framework GameUtil2D. Confira a tabela abaixo:

2.1) O pacote gameutil2d.classes.basic

Nesse pacote estão todas as classes básicas para o desenvolvimento do nosso jogo, voltada para a exibição de imagens, construção de animações, criação de pontos de colisão e etc.

Vejamos agora as classes e seus respectivos métodos desse pacote:

2.1.1) A classe GameElement

Essa uma classe genérica, utilizada como classe base para a construção das outras classes presentes dentro desse e outros pacotes deste framework. Vejamos os seus métodos na tabela abaixo:

Método	Descrição
`public void SetX(int value)`	Método responsável por definir a posição do elemento na coordenada X da tela.
`public void SetY(int value)`	Método responsável por definir a posição do elemento na coordenada Y da tela.
`public void SetWidth(int value)`	Método responsável por definir a largura do elemento.
`public void SetHeight(int value)`	Método responsável por definir a altura do elemento.
`public void SetBounds(int x, int y, int width, int height)`	Método responsável por definir a coordenada X, a coordenada Y, a largura e a altura do elemento.
`public void MoveByX(int value)`	Desloca um elemento pela coordenada X da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro “value” for positivo, desloca o elemento para direita, caso contrário, move para esquerda.

Método	Descrição
`public void MoveByY(int value)`	Desloca um elemento pela coordenada Y da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para baixo, caso contrário, move para cima.
`public void SetTag(String tag)`	Método responsável por definir uma tag "rotulo" para o nosso objeto (muito útil quando trabalhado com a classe Character e Scene).
`public String GetTag()`	Método responsável por retornar a "tag" definida para o elemento do jogo.
`public int GetX()`	Método responsável por retornar a coordenada X do elemento na tela do jogo.
`public int GetY()`	Método responsável por retornar a coordenada Y do elemento na tela do jogo.
`public int GetWidth()`	Método responsável por retornar largura do elemento do jogo.
`public int GetHeight()`	Método responsável por retornar altura do elemento do jogo.
`public boolean IsClicked(int x, int y)`	Método que retorna verdadeiro se o elemento foi clicado na tela, baseado nas coordenadas X e Y onde ocorreu o clique.
`public void Draw(Graphics graphics)`	Método abstrato (implementado nas classes derivadas como Image, AnimationSprites e Character) responsável por desenhar o elemento na tela

2.1.2) A classe Image

A classe **Image** é responsável por exibir as imagens (sprites e texturas) na tela do seu jogo. Ela é derivada da classe **GameElement** (ou seja, possui todos os métodos da classe **GameElement**).

Vejamos na tabela abaixo seus métodos:

Método	Descrição
`public Image(String image_name, int x, int y, int width, int height)`	Construtor da classe responsável por carregar a imagem na memória. No parâmetro do construtor informamos um argumento do tipo o nome da imagem, as coordenadas do elemento (parâmetros x e y), a largura (width) e a altura (height).

Método	Descrição
`public void SetX(int value)`	Método responsável por definir a posição do elemento na coordenada X da tela.
`public void SetY(int value)`	Método responsável por definir a posição do elemento na coordenada Y da tela.
`public void SetWidth(int value)`	Método responsável por definir a largura do elemento.
`public void SetHeight(int value)`	Método responsável por definir a altura do elemento.
`public void SetBounds(int x, int y, int width, int height)`	Método responsável por definir a coordenada X, a coordenada Y, a largura e a altura do elemento.
`public void MoveByX(int value)`	Desloca um elemento pela coordenada X da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para direita, caso contrário, move para esquerda.
`public void MoveByY(int value)`	Desloca um elemento pela coordenada Y da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para baixo, caso contrário, move para cima.
`public void SetTag(String tag)`	Método responsável por definir uma tag "rotulo" para o nosso objeto (muito útil quando trabalhado com a classe Character e Scene).
`public String GetTag()`	Método responsável por retornar a "tag" definida para o elemento do jogo.
`public int GetX()`	Método responsável por retornar a coordenada X do elemento na tela do jogo.
`public int GetY()`	Método responsável por retornar a coordenada Y do elemento na tela do jogo.
`public int GetWidth()`	Método responsável por retornar largura do elemento do jogo.
`public int GetHeight()`	Método responsável por retornar altura do elemento do jogo.
`public boolean IsClicked(int x, int y)`	Método que retorna verdadeiro se o elemento foi clicado na tela, baseado nas coordenadas X e Y onde ocorreu o clique.
`public void Draw(Graphics graphics, [FlipEffect flipEffect])`	Método responsável por desenhar a imagem na tela, com efeito de Flip (inversão). Se o parâmetro "effect" (**que é opcional**) possuir o valor "FlipEffect.HORIZONTAL" (ou true), inverte a imagem na horizontal.

2.1.3) A classe AnimationSprites

A classe **AnimationSprites** é responsável por exibir uma animação de sprites dentro de um jogo. Ela é derivada da classe **GameElement**:

Vejamos na tabela abaixo seus métodos:

Método	Descrição
`public AnimationSprites(int x, int y, int width, int height)`	Construtor da classe responsável por carregar o objeto que ira exibir a animação. No parâmetro do construtor informamos as coordenadas da animação (parâmetros x e y), a largura (w) e a altura (y) das sprites que estarão no objeto.
`public void Add(String image_name)`	Método responsável por adicionar uma imagem dentro do objeto, que fará parte da animação de sprites.
`public void Start(int frames, boolean loop)`	Método que inicia a animação de sprites. Nesse método definimos o intervalo de troca das imagens em frames (no parâmetro "frames"), e se a animação ocorrerá em loop (caso o parâmetro "loop" seja true) ou não (caso o parâmetro "loop" seja false)
`public void Stop()`	Método que encerra a execução da animação.
`public void Draw(Graphics graphics, [FlipEffect flipEffect])`	Método responsável por desenhar a animação na tela, com efeito de Flip (inversão). Se o parâmetro "effect" (que é opcional) possuir o valor "FlipEffect.HORIZONTAL" (ou true), inverte a animação na horizontal.
`public void SetX(int value)`	Método responsável por definir a posição do elemento na coordenada X da tela.
`public void SetY(int value)`	Método responsável por definir a posição do elemento na coordenada Y da tela.
`public void SetWidth(int value)`	Método responsável por definir a largura do elemento.
`public void SetHeight(int value)`	Método responsável por definir a altura do elemento.
`public void SetBounds(int x, int y, int width, int height)`	Método responsável por definir a coordenada X, a coordenada Y, a largura e a altura do elemento.
`public void MoveByX(int value)`	Desloca um elemento pela coordenada X da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para direita, caso contrário, move para esquerda.

Método	Descrição
`public void MoveByY(int value)`	Desloca um elemento pela coordenada Y da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para baixo, caso contrário, move para cima.
`public void SetTag(String tag)`	Método responsável por definir uma tag "rotulo" para o nosso objeto (muito útil quando trabalhado com a classe Character e Scene).
`public String GetTag()`	Método responsável por retornar a "tag" definida para o elemento do jogo.
`public int GetX()`	Método responsável por retornar a coordenada X do elemento na tela do jogo.
`public int GetY()`	Método responsável por retornar a coordenada Y do elemento na tela do jogo.
`public int GetWidth()`	Método responsável por retornar largura do elemento do jogo.
`public int GetHeight()`	Método responsável por retornar altura do elemento do jogo.
`public boolean IsClicked(int x, int y)`	Método que retorna verdadeiro se o elemento foi clicado na tela, baseado nas coordenadas X e Y onde ocorreu o clique.

2.1.4) A classe Collision

Essa classe é responsável por verificar a colisão entre objetos dentro da tela do jogo. Vejamos seus métodos abaixo:

Método	Descrição
`public boolean Check(GameElement obj1, GameElement obj2)`	Método responsável verificar se houve uma colisão entre dois objetos. Retorna "true" quando ocorrer uma colisão, e falso caso contrário.
`public boolean Check(GameElement obj, Scene scene, String any_object_with_tag)`	Método responsável por verificar se houve a colisão de um objeto com algum outro elemento qualquer dentro de uma cena (ver sobre a classe Scene mais para frente), que possua uma tag especificada no parâmetro (any_object_width_tag). O método retorna "true" se ocorreu uma colisão, e falso caso contrário.

Método	Descrição
`public GameElement CheckAndReturn (GameElement obj, Scene scene, String any_object_with_tag)`	Verifica se houve a colisão de um objeto com algum outro elemento qualquer dentro de uma cena que possua uma tag especificada no parâmetro (any_object_width_tag). Diferente do método "Check", esse retorna a instância do elemento que colidiu com o objeto (se uma colisão acontecer). Caso nenhuma colisão tenha acontecido, o método retorna **null**.

2.1.5) A classe Box

Essa classe (derivada da classe **GameElement**), possui somente uma única finalidade: definir pontos de colisão. Veja seus métodos abaixo:

Método	Descrição
`public Box(int x, int y, int width, int height, [String tag])`	Construtor da classe responsável por carregar os pontos de colisão, onde informamos as coordenadas X e Y, como também a largura, altura e sua tag (que é opcional).

2.2) O pacote gameutil2d.classes.character

Nesse pacote existe somente uma classe, destinada para a criação de personagens (para jogos no estilo plataforma). Vamos conhecer a classe:

2.2.1) A classe Character

Com a classe **Character** podemos criar personagens voltados para jogos no estilo plataforma. A classe já possui sistema de física (pulo e queda), detecção de colisões e etc. Vejamos seus métodos abaixo:

Método	Descrição
`public Character(int x, int y, int width, int height)`	Construtor da classe responsável por carregar o objeto que irá exibir o personagem. No parâmetro do construtor informamos as coordenadas da animação (parâmetros x e y), a largura (width) e a altura (height) das sprites que estarão no objeto.
`public void AddNewSpritesIdle(String animation_name, string...image_names)`	Método responsável por adicionar as sprites que representam o estado de repouso de um personagem (quando o ele fica parado em movimento). Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer a aquela animação.
`public void AddNewSpritesMove(String animation_name, string...image_names)`	Método responsável por adicionar as sprites que representam o movimento do personagem (que pode ser dele correndo ou andando). Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.
`public void AddNewSpritesAttack(String animation_name, string...image_names)`	Método responsável por adicionar as sprites que representam o ataque do personagem (como um soco, chute ou qualquer outro golpe). Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.
`public void AddNewSpritesJumping(String animation_name, string...image_names)`	Método responsável por adicionar as sprites que representam o pulo do personagem. Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.
`public void AddNewSpritesDamage(String animation_name, string...image_names)`	Método responsável por adicionar as sprites que representam o personagem sofrendo algum dano. Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.

Método	Descrição
public void AddNewSpritesMove(String animation_name, string...image_names)	Método responsável por adicionar as sprites que representam o personagem morrendo. Cada animação deve possuir um nome e o conjunto de imagens que vão pertencer aquela animação.
public void Idle(String animation_name, int frames, boolean loop)	Executa a animação do personagem em estado de repouso (parado). Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
public void Idle(int frames, boolean loop)	Executa a animação do personagem em estado de repouso (parado). Como nesse método não informamos nenhum nome, será executado a primeira animação de repouso que estiver dentro do personagem. Os únicos argumentos que informamos é o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
public void MoveToRight(String animation_name,int frames, boolean loop)	Executa a animação do personagem se movendo para a direita. Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
public void MoveToRight(int frames, boolean loop)	Executa a animação do personagem se movendo para a direita. Como nesse método não informamos nenhum nome, será executado a primeira animação de movimento que estiver dentro do personagem. Os únicos argumentos que informamos é o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
public void MoveToLeft(String animation_name,int frames, boolean loop)	Executa a animação do personagem se movendo para a esquerda. Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
public void MoveToLeft(int frames, boolean loop)	Executa a animação do personagem se movendo para a esquerda. Como nesse método não informamos nenhum nome, será executado a primeira animação de movimento que estiver dentro do personagem. Os únicos argumentos que informamos é o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).

Método	Descrição
`public void Attack(String animation_name, int frames, boolean loop)`	Executa a animação do personagem atacando. Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
`public void Attack(int frames, boolean loop)`	Executa a animação do personagem atacando. Como nesse método não informamos nenhum nome, será executado a primeira animação de ataque que estiver dentro do personagem. Os únicos argumentos que informamos é o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
`public void Jump(String animation_name,int frames, boolean loop)`	Executa a animação do personagem pulando. Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
`public void Jump(int frames, boolean loop)`	Executa a animação do personagem atacando. Como nesse método não informamos nenhum nome, será executado a primeira animação de ataque que estiver dentro do personagem. Os únicos argumentos que informamos é o intervalo da troca das sprites (em frames) e se a animação ficará em loop (true) ou não (false).
`public void Jump(String animation_name,int frames, int jump_shift, boolean loop)`	Executa a animação do personagem pulando. Nesse método devemos informar o nome da animação, o intervalo da troca das sprites (em frames), o valor de deslocamento do pulo do personagem e se a animação ficará em loop (true) ou não (false).
`public void Jump(int frames, int jump_shift, boolean loop)`	Executa a animação do personagem atacando. Como nesse método não informamos nenhum nome, será executado a primeira animação de ataque que estiver dentro do personagem. Os únicos argumentos que informamos é o intervalo da troca das sprites (em frames) , o valor de deslocamento do pulo do personagem e se a animação ficará em loop (true) ou não (false).

Método	Descrição
`public void SufferDamage(String animation_name, int frames)`	Executa a animação do personagem sofrendo dano (além de mostrar o personagem piscando na tela). Nesse método devemos informar o nome da animação e o intervalo da troca das sprites (em frames).
`public void SufferDamage(int frames)`	Executa a animação do personagem sofrendo dano (além de mostrar o personagem piscando na tela). Como nesse método não informamos nenhum nome, será executado a primeira animação do personagem sofrendo dano presente dentro do objeto. O único argumento que informamos é o intervalo da troca das sprites (em frames).
`public void Die(String animation_name, int frames)`	Executa a animação do personagem morrendo. Nesse método devemos informar o nome da animação e o intervalo da troca das sprites (em frames).
`public void Die(int frames)`	Executa a animação do personagem morrendo. Como nesse método não informamos nenhum nome, será executado a primeira animação do personagem morrendo presente dentro do objeto. O único argumento que informamos é o intervalo da troca das sprites (em frames).
`public boolean IsAttacking()`	Método que retorna verdadeiro se o personagem está atacando, caso contrário, retorna falso.
`public boolean IsDamaged()`	Método que retorna verdadeiro se o personagem está sofrendo dano, caso contrário, retorna falso.
`public boolean IsDying()`	Método que retorna verdadeiro se o personagem está morrendo (quando a animação dele morrendo está em execução), caso contrário, retorna falso.
`public boolean IsDead()`	Método que retorna verdadeiro se o personagem está morto (quando a animação dele morrendo é encerrada), caso contrário, retorna falso.
`public boolean IsGround()`	Método que retorna verdadeiro se o personagem está no chão, caso contrário, retorna falso.
`public void SetMaxFrameDamageDuration(int frames)`	Nesse método definimos em frames o tempo de duração do dano do personagem.

Método	Descrição
`public boolean CollisionbySide(Scene scene)`	Método que retorna verdadeiro se o personagem sofreu uma colisão pelos lados com objetos que são definidos dentro de um cenário.
`public void AddCollisionElementOfSideByTag(String tag)`	Nesse método definimos os elementos (objetos) de colisão, que o nosso personagem poderá se colidir durante seu movimento pelos lados. Os elementos de colisão são representados por “tags” (e que deverão ser definidos pelo método “SetTag”) e deverão estar presentes dentro de um objeto do tipo Scene, onde o personagem estará adicionado.
`public void AddCollisionElementOfFallByTag(String tag)`	Nesse método definimos os elementos (objetos) de colisão, que o nosso personagem poderá se colidir durante uma queda após o pulo. Os elementos de colisão são representados por “tags” (e que deverão ser definidos pelo método “SetTag”) e deverão estar presentes dentro de um objeto do tipo Scene, onde o personagem estará adicionado.
`public void Update(Scene scene)`	Método responsável por processar o pulo, a queda, as colisões, o ataque e toda a física do personagem.
`public void SetEnableJumpAndFall(boolean jump_fall)`	Quando o método Update está em uso, o mesmo processa o pulo e a queda do personagem. Esse método habilita ou desabilita o pulo e queda do personagem.
`public void TurnToRight()`	Esse método vira nosso personagem para a direita (inverte as imagens), se o mesmo estiver em sentido contrário.
`public void TurnToLeft()`	Esse método vira nosso personagem para a esquerda (inverte as imagens), se o mesmo estiver em sentido contrário.
`public void InvertSprites()`	Inverte todas as sprites do personagem (realiza um efeito de flip na horizontal).
`public void SetX(int value)`	Método responsável por definir a posição do elemento na coordenada X da tela.
`public void SetY(int value)`	Método responsável por definir a posição do elemento na coordenada Y da tela.
`public void SetWidth(int value)`	Método responsável por definir a largura do elemento.

Método	Descrição
`public void SetBounds(int x, int y, int width, int height)`	Método responsável por definir a coordenada X, a coordenada Y, a largura e a altura do elemento.
`public void ShiftSceneElementsOnUpdate(boolean can_shift)`	Esse método permite habilitar o deslocamento de cenário quando o personagem pular (ao invés do personagem se deslocar), bastando definir o parâmetro *can_shift* como **true**.
`public void SetFixedElementsByTag(String…tag_names)`	Usado em conjunto com o método *ShiftSceneElementsOnUpdate* define quais elementos NÃO SERÃO deslocados quando o personagem pular pela sua tag.
`public void MoveByX(int value)`	Desloca um elemento pela coordenada X da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para direita, caso contrário, move para esquerda.
`public void MoveByY(int value)`	Desloca um elemento pela coordenada Y da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o elemento para baixo, caso contrário, move para cima.
`public void SetTag(String tag)`	Método responsável por definir uma tag "rotulo" para o nosso objeto (muito útil quando trabalhado com a classe Character e Scene).
`public String GetTag()`	Método responsável por retornar a "tag" definida para o elemento do jogo.
`public int GetX()`	Método responsável por retornar a coordenada X do elemento na tela do jogo.
`public int GetY()`	Método responsável por retornar a coordenada Y do elemento na tela do jogo.
`public int GetWidth()`	Método responsável por retornar largura do elemento do jogo.
`public int GetHeight()`	Método responsável por retornar altura do elemento do jogo.
`public boolean IsClicked(int x, int y)`	Método que retorna verdadeiro se o elemento foi clicado na tela, baseado nas coordenadas X e Y onde ocorreu o clique.

2.3) O pacote gameutil2d.classes.scene

Nesse pacote existe somente uma classe, destinada para a criação de cenários do jogo.

2.3.1) A classe Scene

Com a classe Scene podemos criar um cenário, onde estarão todos os elementos que serão visualizados no jogo. Seu uso é opcional, mas, quando for utilizar a classe Character para a construção de personagens, seu uso torna-se (teoricamente) obrigatório. Vejamos os seus métodos.

Método	Descrição
`public Scene()`	Construtor da classe Scene.
`public void Add(GameElement element)`	Método que adiciona um elemento dentro do objeto scene.
`public GameElement Get(int index)`	Método que retorna um objeto (do tipo GameElement) presente em um determinado índice (especificado no parâmetro index).
`public void Remove(int index)`	Remove um elemento do objeto scene, baseado em seu índice.
`public void Remove(GameElement element)`	Remove um elemento do objeto scene, baseado em sua instância.
`public int GetCount()`	Retorna o total de elementos dentro do objeto.
`public int GetCountByType(string type)`	Conta e retorna o total de elementos dentro do objeto de um determinado tipo de dados, informada no parâmetro “type”.
`public int GetCountByTag(string tag)`	Conta e retorna o total de elementos dentro do objeto que possuem uma determinada tag, informada no parâmetro “tag”.
`public ArrayList<GameElement>Elements()`	Método que retorna um array, que contém todos os elementos presentes dentro do objeto scene.
`public void MoveByX(int value)`	Desloca todos os elementos dentro do objeto scene pela coordenada X da tela (partindo da posição de origem de cada um).
`public void MoveByY(int value)`	Desloca todos os elementos dentro do objeto scene pela coordenada Y da tela (partindo da posição de origem de cada um).

2.4) O pacote gameutil2d.classes.text

Nesse pacote existe somente uma classe, destinada para exibir textos (muito utilizado em HUDs.)

2.4.1) A classe TextDrawable

Com a classe **TextDrawable** podemos exibir uma informação na tela do jogo (ideal para a construção de uma HUD (Head-Up Display)). Vejamos seus métodos a seguir:

Método	Descrição
`public TextDrawable()`	Construtor da classe TextDrawable.
`public void SetSize(int size)`	Definimos o tamanho do texto a ser exibido na tela.
`public void SetColor(int color)`	Definimos a cor do texto a ser exibido na tela
`public void SetX(int value)`	Define a posição na coordenada X onde iremos exibir o texto.
`public void SetY(int value)`	Define a posição na coordenada Y onde iremos exibir o texto.
`public void SetXY(int x, int y)`	Define a posição nas coordenadas X e Y onde iremos exibir o texto.
`public void DrawString(Graphics graphics, String text)`	Exibe seu texto na tela do jogo.
`public void MoveByX(int value)`	Desloca o texto pela coordenada X da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o texto para direita, caso contrário, move para esquerda.
`public void MoveByY(int value)`	Desloca o texto pela coordenada Y da tela (partindo a posição de origem do mesmo). Se o valor informado no parâmetro "value" for positivo, desloca o texto para baixo, caso contrário, move para cima.

2.5) O pacote gameutil2d.classes.media

Nesse pacote estão todas as classes responsáveis por reproduzir músicas e sons de efeito em um jogo. Vejamos as suas classes:

2.5.1) A classe Song

Essa classe só possui uma única finalidade: Armazenar a música que será reproduzida (através da classe **MediaPlayer**) em um jogo. Vejamos o seu método.

Método	Descrição
`public Song(String song_name)`	Construtor da classe, que armazena a música a ser reproduzida (que deve estar na pasta "sounds" do seu projeto e no formato "mp3").

2.5.2) A classe MediaPlayer

Essa classe é responsável por reproduzir as músicas (armazenadas dentro de uma instancia do tipo **Song**) dentro de um jogo. Vejamos seus métodos.

Método	Descrição
`public static void Play(Song song)`	Reproduz a música armazenada dentro de uma instância do tipo Song.
`public static void Stop()`	Encerra uma música em execução.
`public static void SetRepeat(boolean repeat)`	Define se a música em execução ficará em loop (true) ou não (false).

2.5.3) A classe SoundEffect

Essa classe é responsável por reproduzir os sons de efeito dentro de um jogo. Vejamos seus métodos.

Método	Descrição
`public SoundEffect(String sound_effect_name)`	Construtor da classe, onde informamos o nome do arquivo de som de efeito (sem extensão). Os arquivos de som devem estar dentro da pasta "sounds" e no formato ".wav".
`public void Play()`	Reproduz o som de efeito no jogo.