# JPlay

# Sumário

# 1 - Criando uma Janela

Para se criar uma janela nós usamos a classe Window do pacote JPlay.

O construtor dessa classe é o seguinte: Window( int width, int height ).

No lugar de '*width*' colocamos o valor da largura da janela em *pixels*.
No lugar de '*height*' colocamos o valor da altura da janela em *pixels*.

Obs.: Toda vez que fomos criar um jogo usando o *JPlay* devemos primeiro criar uma janela, e depois criar todos os outros componentes do jogo, isso é obrigatório, senão for feito o jogo não executará.

**Exemplo 1: Cria e exibe uma janela, com cor cinza e sem bordas.**

```
import JPlay.Window;

public class Main
{
  public static void main(String[] args)
  {
        Window janela = new Window(800,600);
  }
}
```

Para sair da tela aperte Alt + F4.

# 2 – Usando uma imagem como imagem de fundo (background)

Para mostrar uma imagem nós precisamos de uma *Window* e de um *loop*.

A *Window* é necessária para mostrar cada atualização feita pelo usuário ou pelo programador na tela em relação as imagens.

O *loop* é o coração do jogo, é nele que serão colocadas todas as condições de interações e as atualizações das imagens na tela.

Para mostrar as atualizações feitas usamos o método *display* da classe *Windows*, que é chamado da seguinte forma: **janela.display().**

**Obs.:** O método *void display*(), sempre deve ser chamado por último quando temos uma lista de objetos a serem desenhados na tela, como apresentando no exemplo abaixo:

**Exemplo 2: Mostra uma imagem como backGround**

```
import JPlay.Window;
public class Main
{
   public static void main(String[] args)
   {
      Window janela = new Window(800,600);
      GameImage backGround = new GameImage("fundo.png");

      while(true)
      {
            backGround.draw();
            janela.display();//Sempre deve ser chamado por último.
      }
   }
}
```

Para criar uma imagem de fundo usamos a classe GameImage.

O seu construtor é da seguinte forma: GameImage(Nome da Imagem);

No exemplo usamos a seguinte imagem "fundo.png" - a imagem usada está na mesma pasta do projeto.

Obs.: É obrigatório colocar a extensão da imagem.

Java só aceita os seguintes formatos de imagem: png, jpeg e gif. Em todos os exemplos são usadas imagens do tipo png.

O método **void draw()** é usado por todas as classes que necessitem desenhar alguma imagem na janela.

# 3 – Usando o teclado - Exemplo de uso

Para usar o teclado temos a classe Keyboard do pacote JPlay. Não é preciso criar um instância do teclado, pois, a classe *Window* já fornece uma instância do mesmo. Logo, todos os objetos do jogo irão usar a mesma instância de teclado.

Para ter acesso a instância de teclado oferecida pela classe Window, usamos o método **Keyboard getKeyboard(),** que retorna uma instância de teclado.

Para acessar os códigos das teclas padrões do teclado procedemos da seguinte forma: digite Keyboard seguido de um ponto final e aparecerão várias opções, escolha a tecla que lhe convier. No exemplo, a escolhida foi a tecla ESC, através do seguinte modo Keyboard.ESCAPE_KEY.

**Exemplo 03: Quando pressionar a tecla ESC sai do jogo**

```
import JPlay.Window;
public class Main
{
   public static void main(String[] args)
   {
      Window janela = new Window(800,600);
      Keyboard keyboard = janela.getKeyboard();
      GameImage backGround = new GameImage("fundo.png");

      boolean executando = true;
      while(executando)
      {
            backGround.draw();
            janela.display();
            if (keyboard.keyDown(Keyboard.ESCAPE_KEY) )
               executando = false;
      }
      janela.exit();
   }
}
```

O método **boolean keyDown()** da classe Keyboard, retorna *true* se a tecla ESC está pressionada, ao contrário, retorna *false*. Como parâmetro deve ser passado o código da tecla.

Repare que adicionamos uma variável to tipo *boolean* ao exemplo. Assim que o usuário apertar a tecla ESC, ela será setada como *false* e o *loop* se encerrará.

O encerramento do *loop*, não faz a tela do jogo desaparecer, ela continuará a ser mostrada até que o método **void exit()** da classe *Window* seja chamado, ele é usado para fechar a tela.

# 4 – Usando o teclado – comportamento das teclas

Na parte 3 dessa apostila vimos um pequeno exemplo de como usar o teclado. Agora, veremos como adicionar novas teclas e mudar o comportamento das teclas já existentes.

### 4.1 – Teclas Padrões

As teclas *defaults* do teclado do JPlay são as seguintes:
DOWN_KEY, ENTER_KEY, ESCAPE_KEY, LEFT_KEY, RIGHT_KEY, SPACE_KEY, UP_KEY.

### 4.2 – Comportamento das teclas

As teclas presentes no teclado do *JPlay* possuem um dos dois comportamento: DETECT_EVERY_PRESS ou DETECT_INITIAL_PRESS_ONLY.

Antes de vermos o que cada comportamento significa, tenha em mente o seguinte: quando uma tecla está pressionada o *Java* dispara um evento de tecla pressionada. Esse é um detalhe importante para entender os comportamentos citados:

DETECT_EVERY_PRESS - a cada iteração do loop o método keyDown() do teclado retorna *true* se a tecla estiver pressionada, ou seja, retorna *true* enquanto a tecla estiver pressionada.

DETECT_INITIAL_PRESS_ONLY – se a tecla tiver esse comportamento, o método keyDown() só irá retornar *true* no momento do pressionamento da tecla, diferentemento do DETECT_EVERY_PRESS que retorna *true* enquanto a tecla estiver pressionada. Isso significa que o método *keyDown*() só irá retornar *true* outra vez quando a tecla for liberada e novamente pressionada.

O comportamento DETECT_EVERY_PRESS pode ser usado para a movimentação de um boneco. O DETECT_INITAL_PRESS_ONLY pode ser usado para fazer um boneco ou nave atirar.

### 4.3 – Adicionando teclas

Para adicionar uma tecla ao teclado do JPlay, você necessita saber qual o código da tecla que será adicionada, para isso use a classe KeyEvent, que está presente no *Java* e ela guarda os códigos de teclas. Para acessar esses códigos digite 'KeyEvent.VK_'.

Para adicionar uma tecla use o método **void addKey(int keyCode)** ou **void addKey(int keyCode, int behavior)** presentes na classe Keyboard.

Em **void addKey(int keyCode, int behavior)** os parâmetros são o código da tecla que se deseja adicionar e o comportamento da mesma.

Exemplo: adicionar a tecla *control* ao teclado do *JPlay*.

```
teclado.addKey( KeyEvent.VK_CONTROL )
```

Usando o método addKey(intkeyCode) adicionamos uma tecla e seu comportamento será DETECT_INITIAL_PRESS_ONLY.

Se você quiser que o comportamento seja DETECT_EVERY_PRESS, use o método addKey(int keyCode, int behavior) do seguinte modo:

```
teclado. addKey(KeyEvent.VK_CONTROL, Keyboard. DETECT_EVERY_PRESS).
```

Obs.: Se for feita a tentativa de adicionar uma tecla já existente, a mesma será substituída por aquela que estiver sendo adicionada, pois, há a possibilidade da nova tecla ter um comportamento diferente da anterior.

### 4.4 – Comportamentos das teclas padrões

As teclas UP_KEY, LEFT_KEY, RIGHT_KEY e DOWN_KEY possuem o comportamento DETECT_EVERY_PRESS.

As teclas ESCAPE_KEY, SPACE_KEY e ENTER_KEY possuem o comportamento DETECT_INITAL_PRESS_ONLY.

### 4.5 – Mudando o comportamento de uma tecla

Para mudar o comportamento de uma tecla utilize o método **void setBehavior(int key, int behavior),** presente na classe *Keyboard* do *JPlay*.

Exemplo: Mudando o comportamento da tecla UP_KEY:

```
teclado.setBehavior(Keyboard.UP_KEY, Keyboard. DETECT_INITIAL_PRESS_ONLY).
```

Os parâmetros a serem passados são: o código da tecla e o novo comportamento.

### 4.6 – Removendo uma tecla

Para remover uma tecla do teclado do JPlay use o método **void removeKey(int key)**.

**Exemplo 4: Adiciona a tecla G que será usada para encerrar o jogo e remove a tecla ESCAPE.**

```
public class Exemplo04
{
  public static void main(String[] args)
  {
    Window janela = new Window(800,600);
    Keyboard keyboard = janela.getKeyboard();
    GameImage backGround = new GameImage("fundo.png");

    keyboard.addKey(KeyEvent.VK_G); //Adiciona a tecla G com o comportamento DETECT_INITIAL_PRESS_ONLY
    keyboard.removeKey(KeyEvent.VK_ESCAPE);

    boolean executando = true;
    while(executando)
    {
        backGround.draw();
        janela.display();
        if ( keyboard.keyDown(KeyEvent.VK_G) )
          executando = false;
    }
    janela.exit();
  }
}
```

# 5 – Mouse

### 5.1 – Usando o mouse

A classe Window fornece uma instância da classe *Mouse*. Para usar a mesma faça do seguinte modo:

*Mouse mouse* = **janela.getMouse();**

O método ***Mouse getMouse*()** existente na classe *Window* retorna uma instância da classe *Mouse*.

Assim como o teclado os botões do mouse também possuem comportamento e estes podem ser mudados. Porém, não é possível adicionar ou retirar botões da classe Mouse.

### 5.2 – Botões padrões do mouse

Eles são os seguintes: BUTTON_LEFT, BUTTON_MIDDLE e BUTTON_RIGHT.
Para acessar o código desses botões faça como se segue: 'Mouse. BUTTON_RIGHT'.
Todos os botões padrões possuem o comportamento DETECT_INITAL_PRESS_ONLY.

### 5.3 – Mudando o comportamento de um botão

Use o método **void setBehavior(int numberBotton, int behavior)**.

Os parâmetros a serem passados são: o código do botão a ser mudado e o comportamento pretendido do botão.

Exemplo: Mudando o comportamento do botão direito:

```
mouse. setBehavior(Mouse. BUTTON_RIGHT, Mouse. DETECT_EVERY_PRESS);
```

Agora, enquanto o botão direito estiver pressionado, o método **boolean isRightButtonPressed()**, descrito abaixo, irá retornar *true*.

### 5.4 – Botões pressionados

Para saber se um botão está pressionado use os seguintes métodos da classe Mouse:

**public boolean isLeftButtonPressed()** – retorna *true* se o botão esquerdo está pressionado, ao contrário, retorna *false*;

**public boolean isMiddleButtonPressed ()** – retorna *true* se o scroll do mouse está pressionado, ao contrário, retorna *false*;

**public boolean isRightButtonPressed** () – retorna *true* se o botão direito do mouse está pressionado, ao contrário, retorna *false*;

### 5.5 – Recuperando a posição do mouse

Para recuperar a posição (x,y) do mouse na tela usamos o método **Point getPosition()**.

Exemplo:

Point posicaoMouse = **mouse.getPosition();**

A classe Point existente no *Java* serve para guardar os valores de (x,y) retornados por *mouse.getPosition()*. Esses valores são acessados da seguinte forma: 'posicaoMouse.x ' e 'posicaoMouse.y'.

### 5.6 – Mouse sobre algum objeto do jogo

Para saber se o mouse está sobre algum objeto do jogo, use o método **boolean isOverObject(GameObject obj)**. Esse método retorna *true* se o mouse estiver sobre o objeto passado como parâmetro, caso contrário, retorna *false*;

O objeto passado como parâmetro pode ser um dos seguintes objetos: GameObject, GameImage, Animation, Sprite ou Body.

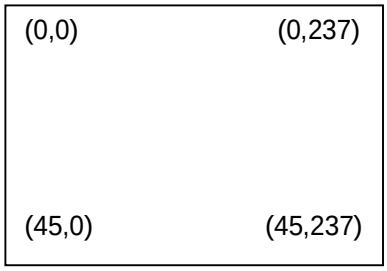
Exemplo:

```
Mouse mouse = janela.getMouse();
Sprite carro = new Sprite("carro.png");

 if (mouse.isOverObject(carro))
    System.out.println("mouse está sobre o carro!")
else
   System.out.println("mouse NÃO está sobre o carro!")
```

### 5.7 – Mouse sobre uma determinada área

Para saber se o mouse está sobre uma determinada área use o **boolean isOverArea( Point start, Point end )** .
Ao lado está representada uma área com 45 pixels de altura e 237 pixels de largura.

Seus pontos mínimo e máximo são (0,0) e (45,237).




Para saber se o mouse está sobre está área faça do seguinte modo:

```
Point pontoMinimo = new Point(0,0);
Point pontoMaximo = new Point(45,237).

if (mouse. isOverArea( pontoMinimo, pontoMaximo ) )
   imprime("Mouse está sobre a área");
else
  imprime("Não está sobre a área!")
```

O método **boolean isOverArea ( int minX, int minY, int maxX, int maxY )**, poderia ser usado no lugar do **boolean isOverArea( Point start, Point end )**, ambas funcionam da mesma maneira.

O uso do método boolean isOverArea ( int minX, int minY, int maxX, int maxY ), seria do seguinte modo:

```
if (mouse. isOverArea(0, 0, 45, 257 ) )
   imprime("Mouse está sobre a área");
```

### Exemplo 05: Ao clicar com o mouse a imagem muda de posição

```
public class Exemplo05
{
   public static void main(String[] args)
   {
      Window janela = new Window(800,600);
      Keyboard keyboard = janela.getKeyboard();
      Mouse mouse = janela.getMouse();
```

```
        GameImage backGround = new GameImage("fundo.png");
        GameImage imagem = new GameImage("megaMan.png");

        boolean executando = true;
        while(executando)
        {
            backGround.draw();
            imagem.draw();
            janela.display();

            if (mouse.isLeftButtonPressed() == true)//Se o mouse clicar muda a posição (x,y) da imagem
               imagem.setPosition( mouse.getPosition() );

            if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true)
               executando = false;
        }
        janela.exit();
    }
}
```

A ordem em que as imagens são desenhas é importante. Se desenharmos primeiro a 'imagem' e depois o backGround, a 'imagem' será sobreposta.

# 6 – Classe GameObject

## 6.1 – A classe

Esta classe é responsável por armazenar as coordenadas (x,y) da imagen e sua dimensão (largura, altura).

Obs.: Toda as classes GameImage, Animation, Sprite e Body são filhas da classe GameObject e possuem as mesmas variáveis e métodos apresentados abaixo:

### 6.2 – Acessando membros da classe

As variáveis x, y, largura, altura, são publicas e podem ser acessadas diretamente do seguinte modo:

```
GameObject objeto = new GameObject();

objecto.x = 100; //Seta a coordenada X do objeto para 100
objecto.y = 385; //Seta a coordenada Y do objeto para 385

objeto.width = 20; //Seta a largura do objeto para 20
objeto.height = 29; //Seta a largura do objeto para 29
```

### 6.3 – Métodos existentes na classe GameObject

Os métodos existentes nessa classe possuem nomes auto-explicáveis e são os seguintes:

**public Point getPosition() -** retorna a posição do GameObject, um ponto no espaço da tela, coordenadas (x,y).

**public Dimension getDimension() -** retorna a dimensão da imagem (largura, altura).

**public void setPosition(int x, int y) -** seta a posição da imagem.

**public void setPosition(Point point) -** seta a posição da imagem.

**public void setDimension(int width, int height) -** seta a dimensão da imagem (largura, altura).

**public void setDimension(Dimension dimension) -** seta a dimensão da imagem.

# 7 – Animação

### 7.1 – Como criar uma animação?

Para criar uma animação precisamos de uma imagem e que ela contenha alguns frames. O número de frames é uma escolha sua.

Um frame é um pedaço da imagem responsável por um movimento da animação.

Exemplo: temos a imagem abaixo:

Repare que existem quatro desenhos do Megaman em uma única imagem. Sendo que cada um desses desenhos pode ser chamado de frame. Logo, essa imagem possui 4 frames.

O conceito de frame é muito importante em animações, tenha-o sempre em mente.

### 7.2 – Instanciando a classe Animation

A classe a Animation tem dois construtores. Por enquanto, o único que nos interessa é o seguinte:

**Animation(nome da imagem, número de frames).**

Para criar um objeto dessa classe procedemos do seguinte modo:

```
Animation animacao = new Animation("animacao01.png", 4);
```

nome da imagem = animacao01.png
número de frames = 4

### 7.3 – Setando o tempo entre a mudança de frames

Em animações os frames devem mudar depois de um certo tempo. Para informar o tempo de mudança entre os frames, ou seja, o tempo em que cada frame será apresentado na tela, usamos o método **void setTimeChangeFrame(long time)**.

```
animação.setTimeChangeFrame(125);
```

Seta o tempo de mudança entre cada um dos 4 frames, isto é, a cada 125 milissegundos o frame apresentado na tela irá mudar, ou falando de outro modo, cada frame será apresentando na tela por 125 milissegundos.

O tempo a ser setado, pelo método apresentado, deve estar em milissegundos.
Lembre-se que 1 segundo é igual a 1000 milissegundos, 1s = 1ms.

### 7.4 – Executando a animação

Para fazer a animação ser executada o método **void runAnimation()** deve ser chamado, ele é o responsável pela troca de frames, respeitando o tempo estipulado pelo método void setTimeChangeFrame(long);

Até agora temos:

```
Animation animacao = new Animation("animacao01.png", 4);
animação.setTimeChangeFrame(125);
animacao.runAnimation();
```

Estamos prontos para criar a nossa primeira animação.

**Exemplo 06: Roda uma animação**

```
public class Exemplo06
{
   public static void main(String[] args)
   {
      Window janela = new Window(800,600);
      Keyboard keyboard = janela.getKeyboard();
      Mouse mouse = janela.getMouse();

      GameImage backGround = new GameImage("fundo.png");
      Animation animacao = new Animation("animacao01.png", 4);

      animacao.setPosition(300, 300);
      animacao.setTimeChangeFrame(125);

      boolean executando = true;
      while(executando)
      {
            backGround.draw();
            animacao.draw();
            janela.display();

            animacao.runAnimation();

            if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true)
               executando = false;
      }
      janela.exit();
   }
}
```

### 7.5 – Executando a animação somente uma vez

Para executar a animação somente uma vez, use o método **void setRepeatAnimation(*boolean*)** da classe Animation. O valor passado por parâmetro deve ser *false*.

```
animação.setRepeatAnimation(false);
```

Quando uma animação é criada, ela será executada indefinidamente, para que isso não ocorra deve-se o usar o método mostrado acima passando o parâmetro *false*.

Se durante o jogo houver a necessidade de que a animação volte a ser executada use o método void setRepeatAnimation(*boolean*), passando como parâmetro o valor *true.*

```
animação.setRepeatAnimation(true);
```

**Exemplo 07: Pausar uma animação**

```
public class Exemplo07
{
   public static void main(String[] args)
   {
      Window janela = new Window(800,600);
      Keyboard keyboard = janela.getKeyboard();

      GameImage backGround = new GameImage("fundo.png");
      Animation animacao = new Animation("animacao01.png", 4);

      animacao.setPosition(300, 300);
      animacao.setTimeChangeFrame(125);
      animacao.setRepeatAnimation(true);

      long time = 0;

      boolean executando = true;
      while(executando)
      {
            backGround.draw();
            animacao.draw();
            janela.display();

            animacao.runAnimation();

            time += janela.timeElapsed();

            if (time > 4000 && time < 5000)
               animacao.setRepeatAnimation(false);
            else
                  if (time > 10000)
                     animacao.setRepeatAnimation(true);

            if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true)
               executando = false;
      }
      janela.exit();
   }
}
```

A variável *time* serve para armazenar a quantidade de tempo.

O comando **long janela.timeElapsed()** retorna a quantidade de tempo em milissegundos passados deste a última atualização da tela e o momento de chamada do método.

Assim o comando time += janela.timeElapsed()  é usado para contar o tempo.

Se o tempo passado for maior do que 4000 milissegundos (4 segundos) e menor do que 5000 milissegundos (5 segundos), faz a animação parar de rodar.

```
if (time > 4000 && time < 5000)
   animacao.setRepeatAnimation(false);
```

Se o tempo passado for maior do que 10 segundos volta a rodar a animação.

```
if (time > 10000)
   animacao.setRepeatAnimation(true);
```

### 7.6 – Trocando os frames manualmente

Para setar quais são os frames a serem usados na animação uso o método
**void setRangeOfFrames(int frameInicial, int frameFinal)**.

Como se está setando os frames manualmente deve-se tomar o cuidado de fazer o seguinte:
animacao.setRangeOfFrames(0, 0),
isso garante que não haverá a troca de frames antes que elas realmente tenham que acontecer. Para saber o que aconteceria se isso não fosse feito experimente apagar esse comando.

**Exemplo 08: Trocando os frames manualmente**

```
public class Exemplo08
{
   public static void main(String[] args)
   {
      Window janela = new Window(800,600);
      Keyboard keyboard = janela.getKeyboard();

      GameImage backGround = new GameImage("fundo.png");
      Animation animacao = new Animation("animacao02.png", 28);

      animacao.setPosition(300, 300);
      animacao.setTimeChangeFrame(80);
      animacao.setRangeOfFrames(0, 0);

      boolean executando = true;
      while(executando)
      {
            backGround.draw();
            animacao.draw();
            janela.display();

            animacao.runAnimation();

            if(keyboard.keyDown(Keyboard.LEFT_KEY))
               animacao.setRangeOfFrames(0, 13);
            else
               if(keyboard.keyDown(Keyboard.RIGHT_KEY))
                  animacao.setRangeOfFrames(14, 27);

            if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true)
               executando = false;
```

```
        }
        janela.exit();
    }
}
```

No trecho de código abaixo temos que se a seta para a esquerda for apertada trocamos os frames que devem ser utilizados.

```
if(keyboard.keyDown(Keyboard.LEFT_KEY))
    animacao.setRangeOfFrames(0, 13);
```

Para entender melhor o que foi dito, pense no seguinte, quando apertar a seta para a esquerda os frames a serem utilizados na animação serão os 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 e 13.

Já para o trecho de código abaixo:

```
if(keyboard.keyDown(Keyboard.RIGHT_KEY))
    animacao.setRangeOfFrames(14, 27);
```

os frames a serem utilizados na animação serão os 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 25, 26, 27.

### 7.7 – Setando um tempo para cada frame

Em certas animações certa parte da animação deve ser executada mais rápida ou mais lenta do que outra parte. Para isso existe o método

```
public void setTimeOfFrame(int frame, long time);
```

Os parâmetros são: o número do frame e o valor do tempo em que o frame deve ser mostrado.

Exemplo:

```
animação. setTimeOfFrame(0, 100); // frame 0 será mostrado por 100 milissegundos
animação. setTimeOfFrame(1, 150); // frame 1 será mostrado por 150 milissegundos
animação. setTimeOfFrame(2, 120); // frame 2 será mostrado por 120 milissegundos
animação. setTimeOfFrame(3, 50);  // frame 3 será mostrado por 50 milissegundos
animação. setTimeOfFrame(4, 50);  // frame 4 será mostrado por 50 milissegundos
animação. setTimeOfFrame(5, 50);  // frame 5 será mostrado por 50 milissegundos
animação. setTimeOfFrame(6, 50);  // frame 6 será mostrado por 50 milissegundos
```

### 7.8 – Escondendo a animação

Para tornar a animação invisível use o método **public void hide().**

Para torna a animação novamente visível use o método **public void unhide();**

Obs.: Mesmo que animação não seja mostrada na tela, se o método runAnimation() estiver sendo chamado, os frames continuarão a ser trocados.

### 7.9 – Últimas considerações

Se o que foi apresentando não suprir as necessidades, pode-se utilizar o método

**public void setCurrFrame(int frame) -** ) este método seta o frame que deve ser desenhado.

Ao usar este método não é aconselhável que se use o método runAnimation() , pois isso poderia gerar alguns efeitos indesejáveis na troca de frames.

**public boolean isAnimationFinished()** – retorna *true* se houve a troca de todos os frames utilizados na animação.

**public int getCurrFrame()** – retorna o número do frame corrente a ser desenhado.

**public boolean getRepeatAnimation()** – retorna *true* se a animação será repetida, caso contrário, *false*.

**public long getTimeOfFrame(int frame)** – retorna o tempo em milissegundos que o frame será mostrado na tela.

**public void reset()** - reseta animação ao seu estado inicial, ou seja, seta o primeiro frame como o frame que será apresentando na tela.

**public void setInitialFrame(int frame)** – seta o frame que será usado para começar a animação.

**public int getInitalFrame()** – retorna o número do frame usado para começar a animação.

**public void setFinalFrame(int frame)** – seta o número do frame que será apresentado por último.

**public int getFinalFrame()** – retorna o número do frame final que é usado na animação.

**public long getTimeChangeFrame()** – se não houve o uso do método setTimeOfFrame(int, long), todos os frames terão o mesmo tempo de apresentação na tela, o método getTimeChangeFrame retorna esse tempo.

**public boolean getRepeatAnimation()** – retorna *true* se a animação irá se repetir indefinidamente.

**public long getTimeOfFrame(int frame)** – retorna o tempo que o frame será mostrado na tela.

A classe *Animation* não possui métodos que façam a animação se mover pela tela. Para isso você pode usar as variáveis públicas ‘x’ e ‘y’.

# 8 – Sprites

Sprites diferentemente de animações possuem métodos que podem fazer a imagem se locomover pela tela.

A classe Sprite extende a classe Animation, isso quer dizer que tudo que fizemos na parte de animação também pode ser aplicado aos Sprites.

### 8.1 – Criando um Sprite

A classe Sprite possui dois construtores:

**public Sprite(String fileName, int numFrames)** - é passado o nome da imagem que será exibida e o número de frames que ela contém.

**public Sprite(String fileName)** - basta passar o nome da imagem a ser exibida, o número de frames automaticamente será 1.

Exemplo:

```
Sprite boneco = new Sprite("megaMan.png", 28);
                    Ou
Sprite bola = new Sprite("bola.png"); // o número de frames setados será 1
```

### 8.2 – Movendo o sprite

Para mover um sprite existem dois métodos:

**public void moveX()** – move o *Sprite* na tela somente no eixo x.
**public void moveY()** – move o *Sprite* na tela somente no eixo y.

Para determinar a velocidade com que o *Sprite* se moverá no eixo X ou no eixo Y pode ser os métodos
**public void setVelocityX(double velocity)**
**public void setVelocityY(double velocity)**
onde o parâmetro é a velocidade com a qual o *Sprite* deverá se deslocar no eixo x ou y.

**public double getVelocityX() –** retorna a velocidade de deslocamento do *Sprite* no eixo X.
**public double getVelocityY() –** retorna a velocidade de deslocamento do *Sprite* no eixo Y.

Obs.: as teclas padrões para fazer o S*prite* se movimentar são as setas direcionais.

Para fazer com que o S*prite* use outras teclas para se movimentar use os seguintes métodos:

**public void moveX(int leftKey, int rightKey)** – passe os códigos das teclas a serem usadas para locomover o *sprite* para a esquerda ou para a direita.

**public void moveY(int upKey, int downKey)** – passe os códigos das teclas a serem usadas para locomover o *sprite* para cima ou para baixo.

A chamado de algum dos métodos abaixo faz o *sprite* se locomover sem que seja preciso que alguma tecla seja pressionada:

**public void moveToUp() –** move o *sprite* para cima.
**public void moveToDown()** – move o *sprite* para baixo.

**public void moveToLeft()** – move o *sprite* para a esquerda.
**public void moveToRight()** – move o *sprite* para a direita.

Obs.: Todos os métodos apresentados não permitem que o *sprite* se locomova para fora da janela.

### 8.3 – Sentido de locomoção do *Sprite*

Para saber em qual sentido o *sprite* está se deslocando no eixo X use o método
**public char getStateOfX()**
que retorna se o *sprite* está se deslocando para a esquerda, para a direita ou se ele não está se locomovendo.

Para saber o sentido do deslocamento no eixo Y, use o método
**public char getStateOfY()**
que retorna se o *sprite* está se deslocando para cima, para baixo ou não está se deslocando.

Ambos os métodos retornam um *char*. Para determinar qual é o sentido, use as variáveis estáticas da classe Sprite: STOP, LEFT, RIGHT, UPWARD, DOWNWARD.

Exemplo:

```
sprite.moveX();
char sentidoX = sprite. getStateOfX()

if (sentidoX ==  Sprite. LEFT)
    System.out.println("esquerda");
else
        if (sentidoX ==  Sprite. RIGHT)
            System.out.println("direita");
        else
            System.out.println("parado");
```

Exemplo:

```
sprite.moveY();
char sentidoY = sprite. getStateOfY();

if (sentidoY ==  Sprite. DOWNWARD)
    System.out.println("descendo");
else
        if (sentidoY ==  Sprite. UPWARD)
            System.out.println("subindo");
        else
            System.out.println("parado");
```

Para saber se o *sprite* não está se movendo utilize: Sprite.STOP.

**Exemplo 09**: Determina o sentido de locomoção do *sprite* no eixo X e seta o frame correspondente a esse sentido.

```
public class Exemplo09
{
  public static void main(String[] args)
  {
    Window janela = new Window(800,600);
    Keyboard keyboard = janela.getKeyboard();

    GameImage backGround = new GameImage("paisagem.png");
```

```
        Sprite sprite = new Sprite("navio.png", 2);

        sprite.setPosition(300, 300);

        boolean executando = true;
        while(executando)
        {
            backGround.draw();
            sprite.draw();
            janela.display();

            sprite.moveX();

            if(sprite.getStateOfX()== Sprite.LEFT)
               sprite.setCurrFrame(0);
            else
               if(sprite.getStateOfX()== Sprite.RIGHT)
                  sprite.setCurrFrame(1);

            if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true)
                executando = false;
        }
        janela.exit();
    }
}
```

### 8.4 – Fazendo o Sprite pular

Antes de fazer o *sprite* pular, devemos informar o valor da coordenada Y que servirá como chão, para isso use o método

**public void setFloor(int floor).**

Para fazer com que o *sprite* pule use o método **public void jump()**. Esse método usa a tecla SPACE para fazer o *sprite* pular.

Exemplo:

```
sprite.setFloor(500);
sprite.jump();  o sprite irá pular e ao cair não irá ultrapassar a coordenada y = 500;
```

Para controlar a velocidade e a altura do pulo use o método **public void setJumpVelocity(double velocity)** onde o parâmetro a ser passar é a velocidade inicial com que o *sprite* irá se deslocar do chão.

Para saber se o *sprite* ainda está executando o pulo use o método **public boolean isJumping()** que retorna *true* se o sprite não retornou ao chão, caso contrário, retorna *false*.

### 8.5 – Mudando a tecla usada no pulo

Por padrão a tecla a ser usada para fazer o boneco pular é a barra de espaço, caso você queira mudar essa tecla use o método

**public void jump(int keyCode)**

que terá como parâmetro o código da tecla que servirá para acionar o pulo.

### 8.6 – Simulando gravidade

Para simular o efeito de gravidade use o método **public void fall();**

Para saber se o sprite tocou o chão use o método **public boolean isOnFloor**(), que retorna *true* se o chão já foi atingido pelo sprite, ao contrário, retorna *false*.

Assim como no pulo, o valor da coordenada Y que servirá como chão deve ser setada antes de usar o método fall();

Para mudar o valor da gravidade use o método **public void setGravity(double gravity).**

**Exemplo 10:** Simulando o efeito de gravidade

```
public class Exemplo10
{
   public static void main(String[] args)
   {
      Window janela = new Window(800,600);
      Keyboard keyboard = janela.getKeyboard();

      GameImage backGround = new GameImage("fundo.png");
      Sprite pedra = new Sprite("pedra.png".png");

      pedra.setPosition(300, 0);
      pedra.setFloor(500);
      pedra.setGravity(0.000098);

      boolean executando = true;
      while(executando)
      {
            backGround.draw();
            pedra.draw();
            janela.display();

           pedra.fall();

            if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true)
               executando = false;
      }
      janela.exit();
   }
}
```

**Exemplo 11:** Troca a tecla usada para o pulo e faz o *sprite* pular

```
public class Exemplo11
{
   public static void main(String[] args)
   {
      Window janela = new Window(800,600);
      Keyboard keyboard = janela.getKeyboard();

     Keyboard.addKey(KeyEvent.VK_CONTROL);
```

```
        GameImage backGround = new GameImage("fundo.png");
        Sprite bola = new Sprite("baseball.png");

        bola. setPosition(300, 400);
        bola. setFloor(500);

        bola. setGravity(0.005);
        bola.setJumpVelocity(1);
        bola.setVelocityX(0.5);

        boolean executando = true;
        while(executando)
        {
              backGround.draw();
              bola.draw();
              janela.display();

              bola.moveX();
              bola.jump(KeyEvent.VK_CONTROL);

                    if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true)
                executando = false;
        }
        janela.exit();
    }
}
```

### 8.7 – Movendo o sprite de um ponto a outro

Para mover o *sprite* de um ponto da tela até outro sem a intervenção do usuário use o método

```
public void moveTo(double x, double y)
```

Onde os valores passados como parâmetros são as coordenadas para onde se deseja que o *sprite* se desloque.

# 9 - Body

A classe Body do JPlay é muito parecida com a classe Sprite. As diferenças se resumem ao modo como o objeto irá se deslocar.

Na classe Sprite quando o usuário parar de pressionar a tecla para locomoção o objeto pára de se locomover. Já na classe Body ao invés do objeto parar, ele continuará se locomovendo por uma certa distância e ao mesmo tempo sua velocidade irá diminuir até se tornar zero, velocidade = 0.

**Exemplo 12: Movendo um Body pela tela.**

```
public class Exemplo12
{
   public static void main(String[] args)
   {
        Window janela = new Window(800,600);
        Keyboard keyboard = janela.getKeyboard();

        GameImage backGround = new GameImage("paisagem.png");
        Body navio = new Body("navio.png", 2);

        navio.setPosition(300, 250);

        boolean executando = true;
        while(executando)
        {
             backGround.draw();
             navio.draw();
             janela.display();

             navio.moveX();

             char estado = navio.getStateOfX();
             if(estado == Sprite.LEFT)
                navio.setCurrFrame(0);
             else
                if(estado == Sprite.RIGHT)
                   navio.setCurrFrame(1);

             janela.delay(10);

             if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true)
                executando = false;
        }
        janela.exit();
   }
}
```

Repare que o exemplo 12 é muito parecido com o exemplo 11. Em termos de uso, não há grandes diferenças entre as classes Body e Sprite.

### 9.1 – Controle de velocidade e aceleração da classe Body

Para controle da velocidade de deslocamento no eixo X ou no eixo Y, usamos os seguintes métodos presentes na classe Body:

**public void setAcceleration(double acceleration)** – seta o valor da aceleração que será usada para aumentar a velocidade do Body em relação ao eixo Y e ao eixo X.

**public void setDeceleration(double deceleration)** – esse método seta o valor da desaceleração que o objeto sofrerá no momento que o usuário deixar de pressionar a tecla de movimentação.

**public void setMaxVelocityX(double maxVelocity)** – seta a máxima velocidade que o *Body* irá atingir no eixo X.

**public void setMaxVelocityY(double maxVelocity)** – seta a máxima velocidade que o *Body* irá atingir no eixo Y.

# 10 – Colisões

Para descobrir se um objeto colidiu com outro você pode usar o método **boolean collided(GameObject)**, ele está presente nas classes Sprite e Body.

O método boolean collided(GameObject), retorna *true* se houve colisão, ao contrário, retorna *false*.

Exemplo:
```
if( barcoAmarelo.collided(barcoVermelho) == true)
        imprime(“colidiram”);
```

Também pode ser feito usando-se a classe estática *Collision*, com o método **boolean collided(GameObject, GameObject)**

Exemplo:
```
if( Collision.collided( barcoAmarelo, barcoVermelho) == true)
        imprime(“colidiram”);
```

A classe Collision pode ser usada para verificar colisão entre quaisquer um dos objetos: GameObject, GameImage, Animation, Sprite e Body.

**Exemplo14: Verificando colisões entre objetos.**

```
public class Exemplo14
{
  //Detectando colisões, ao colidir aciona a animação da explosão
  public static void main(String[] args)
  {
      Window janela = new Window(800,600);
      Keyboard keyboard = janela.getKeyboard();

      keyboard.addKey(KeyEvent.VK_A, Keyboard.DETECT_EVERY_PRESS);
      keyboard.addKey(KeyEvent.VK_D, Keyboard.DETECT_EVERY_PRESS);

      GameImage backGround = new GameImage("fundo.png");
      Body navioAbobora = new Body("navio.png", 2);
      Body navioAmarelo = new Body("navio2.png",2);
      Animation explosao = new Animation("explosao.png",20);

      explosao.setRepeatAnimation(false);

      navioAbobora.setPosition(500,250);
      navioAmarelo.setPosition(100,250);
      navioAmarelo.setMaxVelocityX(3);
      navioAbobora.setMaxVelocityX(3);

      boolean executando = true;
      boolean acionarExplosao = false;
      while(executando)
      {
          backGround.draw();
          if (acionarExplosao == false)
          {
              navioAbobora.draw();
```

```
            navioAmarelo.draw();

            navioAbobora.moveX();

             //O navio amarelo usará as teclas a,d para se movimentar para os lados
            navioAmarelo.moveX(KeyEvent.VK_A, KeyEvent.VK_D);

            char estado = navioAbobora.getStateOfX();
            if(estado == Sprite.LEFT)
               navioAbobora.setCurrFrame(0);
            else
               if(estado == Sprite.RIGHT)
                  navioAbobora.setCurrFrame(1);

            estado = navioAmarelo.getStateOfX();
            if(estado == Sprite.LEFT)
               navioAmarelo.setCurrFrame(0);
            else
               if(estado == Sprite.RIGHT)
                  navioAmarelo.setCurrFrame(1);

            if (navioAbobora.collided(navioAmarelo))
            {
                  acionarExplosao = true;
                  //O explosao.setPosition(x,y) - serve para centralizar a explosão
                  //entre os navios
                  if(navioAbobora.getStateOfX() == Body.LEFT)
                     explosao.setPosition( navioAbobora.x - navioAbobora.width - explosao.width/4, 150 );
                  else
                     if(navioAmarelo.getStateOfX() == Body.RIGHT)
                        explosao.setPosition( navioAmarelo.x + navioAmarelo.width - explosao.width/2, 150 );
            }
         }
         else
         {
            explosao.runAnimation();
            explosao.draw();
         }
         janela.display();


          if (explosao.isAnimationFinished())
         {
            explosao.hide();
         }

         if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true )
            executando = false;
      }
      janela.exit();
   }
}
```

# 11 – Botão

Para criar um botão usamos o Mouse em conjunto com qualquer uma das seguintes classes: GameObject, GameImage, Animation, Sprite ou Body.

### 11.1 – O processo de click

Para sabermos se o mouse clicou em algum objeto que estamos usando como botão, devemos saber se o mouse está sobre o objeto, para isso usamos o método

**boolean mouse.isOverObject(Objeto)**

Agora que sabemos como verificar se o mouse está sobre um objeto, devemos saber se algum botão foi pressionado, para isso podemos usar os seguintes métodos:

**public boolean isLeftButtonPressed(),**
**public boolean isMiddleButtonPressed(),**
**public boolean isRightButtonPressed().**

Esses dois métodos em conjunto podem fazer com que qualquer objeto tenha a função de um botão.

**Exemplo15: Criando um botão.**

```
public class Exemplo15
{
   //Criando um botão
   public static void main(String[] args)
   {
        Window janela = new Window(800,600);
        Keyboard keyboard = janela.getKeyboard();
        Mouse mouse = janela.getMouse();

        GameImage backGround = new GameImage("fundo.png");

        //Animação a ser usada, ao clicar no botão a animação irá começar a rodar
        Animation botao = new Animation("botao.png",12);

        botao.setPosition(300,300);
        botao.setTimeChangeFrame(100);

        boolean executando = true;
        boolean clicouNoBotao = false;
        while(executando)
        {
             backGround.draw();
             botao.draw();
             janela.display();

             //Primeiro verifica se o mouse está sobre o objeto
             //Depois se o botão esquerdo do mouse clicou
             if (mouse.isOverObject(botao) && mouse.isLeftButtonPressed())
                clicouNoBotao = true;

             if (clicouNoBotao)
```

```
                botao.runAnimation();

            if (botao.isAnimationFinished())
            {
                clicouNoBotao = false;

                //Reseta a animação ao seu estado inicial
                botao.reset();
            }

            if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true )
                executando = false;
        }
        janela.exit();
    }
}
```

# 12 – Som

O JPlay, por enquanto, só aceita arquivos wav.

### 12.1 – Executando um som

Para executar um som, basta fazer:

```
new Sound( nome do arquivo ).play();
```

Esse comando pode ser chamado em qualquer parte do código.

### 12.2 – Repetindo um som

Muitas vezes desejamos que um mesmo som seja executado várias vezes, por exemplo, quando temos uma música que fica tocando em backGround. Para isso bastar usar o método abaixo passando como parâmetro o valor *true*.

```
public void setRepeat(boolean value).
```

### 12.3 - Configurações do som

No pacote do JPlay temos as seguintes opções para lidar com som:

**public boolean isExecuting()** – retorna *true* se o som estiver sendo reproduzido, ao contrário, retorna *false*.
**public void pause()** – pausa o som.
**public void stop()** – pára de tocar o som.
**public void play()** – começa a tocar o som.
**public void setVolume(float value)** – seta o volume com que o som irá ser tocado.
**public void decreaseVolume(float value)** - diminui o volume do som .
**public void increaseVolume(float value)** – aumenta o valor do som.

**Exemplo 16: Ao detectar que dois objetos colidiram reproduz um som de explosão.**

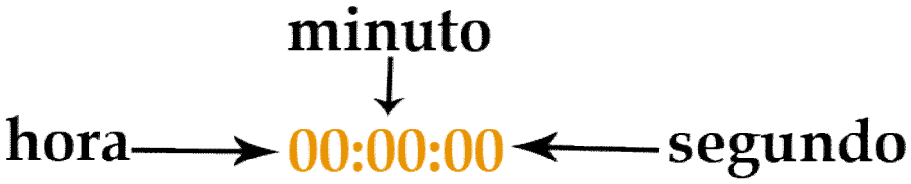
```
public class Exemplo16
{
  //Detectando colisões, ao colidir aciona a animação da explosão
  public static void main(String[] args)
  {
      Window janela = new Window(800,600);
      Keyboard keyboard = janela.getKeyboard();

      keyboard.addKey(KeyEvent.VK_A, Keyboard.DETECT_EVERY_PRESS);
      keyboard.addKey(KeyEvent.VK_D, Keyboard.DETECT_EVERY_PRESS);

      GameImage backGround = new GameImage("fundo.png");
      Body navioAbobora = new Body("navio.png", 2);
      Body navioAmarelo = new Body("navio2.png",2);
      Animation explosao = new Animation("explosao.png",20);

      explosao.setRepeatAnimation(false);

      navioAbobora.setPosition(500,250);
      navioAmarelo.setPosition(100,250);
      navioAmarelo.setMaxVelocityX(3);
```

```
navioAbobora.setMaxVelocityX(3);

boolean executando = true;
boolean acionarExplosao = false;
while(executando)
{
      backGround.draw();
      if (acionarExplosao == false)
      {
            navioAbobora.draw();
            navioAmarelo.draw();

            navioAbobora.moveX();

             //O navio amarelo usará as teclas a,d para se movimentar para os lados
            navioAmarelo.moveX(KeyEvent.VK_A, KeyEvent.VK_D);

            char estado = navioAbobora.getStateOfX();
            if(estado == Sprite.LEFT)
               navioAbobora.setCurrFrame(0);
            else
               if(estado == Sprite.RIGHT)
                  navioAbobora.setCurrFrame(1);

            estado = navioAmarelo.getStateOfX();
            if(estado == Sprite.LEFT)
               navioAmarelo.setCurrFrame(0);
            else
               if(estado == Sprite.RIGHT)
                  navioAmarelo.setCurrFrame(1);

            if (navioAbobora.collided(navioAmarelo))
            {
                  new Sound("explosao.wav").play();
                  acionarExplosao = true;
                  //O explosao.setPosition(x,y) - serve para centralizar a explosão
                  //entre os navios
                  if(navioAbobora.getStateOfX() == Body.LEFT)
                     explosao.setPosition( navioAbobora.x - navioAbobora.width - explosao.width/4, 150 );
                  else
                     if(navioAmarelo.getStateOfX() == Body.RIGHT)
                        explosao.setPosition( navioAmarelo.x + navioAmarelo.width - explosao.width/2, 150 );
            }
      }
      else
      {
         explosao.runAnimation();
         explosao.draw();
      }
      janela.display();


       if (explosao.isAnimationFinished())
      {
         explosao.hide();
      }
```

```
            if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true )
                executando = false;
        }
        janela.exit();
    }
}
```

# 13 - Tempo

No pacote do JPlay temos um contador de tempo no seguinte formato:




### 13.1 – Cronômetro regressivo ou cronômetro progressivo

Para a classe time funcionar como um cronômetro regressivo na opção, boolean crescentTime, passe o valor *false.* Para funcionar como um cronômetro progressivo passe o valor *true.*

Essa opção existe em todos os construtores na classe Time.

### 13.2 - Construtores da classe Time

#### 13.2.1 - public Time(int x, int y, boolean crescentTime)

Para esse construtor devem ser passadas as posições (x,y) de onde o tempo será desenhado e se o Time será usado como um cronômetro regressivo ou progressivo.

Usando esse construtor o tempo é iniciado com o valor zero, ou seja, hora, minutos e segundos são iniciados com o valor zero.

Exemplo:

```
Time tempo = new Time(100,100, true);
X = 100, Y = 100, Cronômetro progressivo.

Time tempo = new Time(100,100, false);
X = 100, Y = 100, Cronômetro regressivo.
```

#### 13.2.2 - public Time( int hour, int minute, int second, int x, int y )

Nesse construtor temos que passar o valor da hora, minuto e segundo que serão usados além da posição onde o tempo será desenhado.

Exemplo: Tempo igual a 5 minutos

```
Time tempo = new Time(0,5,0, 456, 329, false);
hora = 0, minuto = 5, segundo = 0, x = 456, y = 329, tempo decrescente ou cronômetro regressivo.
```

#### 13.2.3 – public Time( int hour, int minute, int second, int x, int y, Font font, Color color, boolean crescentTime )

Nesse construtor temos a liberdade de escolher a cor e a fonte que serão usadas para exibir o tempo.

Exemplo:

```
//public Time( int hour, int minute, int second, int x, int y, Font font, Color color, boolean crescentTime )
Font fonte = new Font("Comic Sans MS", Font. TRUETYPE_FONT, 40);
```

Time tempo = new Time( 1, 23, 34, 100, 100, fonte , Color.Yellow, false);

Obs.: Nos outros construtores a fonte e a cor padrões são as seguintes:
Font("Arial",Font.TRUETYPE_FONT, 20) e Color.YELLOW .

### 13.3 – Métodos da classe Time

**public String toString()** – retorna uma *string* com o valor do tempo, a string retornada é no formato 00:00:00.

**public void draw(String string)** – desenha uma mensagem que é uma escolha do usuário e em seguida desenha o valor do tempo.

**public void draw()** – desenha o valor do tempo na tela.

**public void setColor(Color color)** – seta a cor a ser usada na fonte que será usada para desenhar o tempo.

**public void setFont(Font font)** – seta a fonte que será usada para desenhar o tempo.

**public boolean timeEnded()** – retorna se o tempo terminou, esse método só serve se o tempo escolhido é do tipo cronômetro regressivo.

**public void setHour( int hour )** – seta o valor da hora.
**public void setMinute( int minute )** – seta o valor dos minutes.
**public void setSecond( int second )** – seta o valor dos segundos.

**public long getHour( )** – retorna o valor da hora.
**public long getMinute()** – retorna o valor do minuto.
**public long getSecond()** – retorna o valor do segundo.

**public long getTotalSecond()** – converte o tempo em segundos e retorna o valor total.

**public void setTime(int hour, int minute, int seconds)** – seta o valor do tempo.

### Exemplo 17: Mostra na tela um contador de tempo regressivo e outro progressivo.

```
public class Exemplo17
{

   public static void main(String[] args)
   {
        Window janela = new Window(800,600);
        Keyboard keyboard = janela.getKeyboard();
        GameImage backGround = new GameImage("fundo.png");

        Time tempo1 = new Time(100, 100,true);
        tempo1.setColor(Color.yellow);

        Time tempo2 = new Time(1, 39, 56, 100, 200,false);
        tempo2.setColor(Color.cyan);

        boolean executando = true;

        while(executando)
```

```
        {
            backGround.draw();
            tempo1.draw("Tempo 1: ");
            tempo2.draw("Tempo 2: ");
            janela.display();

            if ( keyboard.keyDown(Keyboard.ESCAPE_KEY) == true )
               executando = false;
        }
        janela.exit();
    }
}
```

# 14 – A Classe Window

Até agora usamos a classe *Window* sem conhecer todos os seus métodos, só utilizamos aqueles que são suficientes para a maioria dos jogos.

O construtor da classe *Window* é o seguinte:

**public Window(int width, int height)** – construtor da classe.

Abaixo há a lista de todos os métodos dessa classe:

**public Keyboard getKeyboard()** – retorna uma instância do teclado.

**public Mouse getMouse()** – retorna uma instância do mouse.

**public Graphics getGameGraphics()** – retorna o *Graphics* onde as imagens são desenhadas.

**public void display()** – mostra a janela atualizada com os desenhos na tela do monitor.

**public void delay(long time)** – aciona uma *Thread* que faz o jogo ficar em modo *sleep* pelo tempo passado por parâmetro, usado principalmente, para atrasar a execução do jogo.

**public long timeElapsed()** – retorna o tempo em milissegundos passado entre a atualização do frame anterior e o atual.

**public void drawText(String message, int x, int y, Color color)** – desenha um texto na tela.

**public void drawText(String message, int x, int y, Color color, Font font)** – desenha um texto na tela.

**public void exit()** – fecha a janela e sai do jogo.

**public Cursor createCustomCursor(String imageName)** – cria um novo cursor do mouse.

**public void clear()** – limpa a tela e a pinta totalmente de preto.