

Classe Console usada em PG

A classe Console permite realizar escritas e leituras numa janela de texto com cores e coordenadas (linha e coluna).

Para usar a classe Console deve ser copiado para o diretório de trabalho o ficheiro jar [ConsolePG.jar](#) e cada ficheiro Java que a use deve fazer **import** `isel.leic.pg.Console`;

Para compilar na linha de comandos um ficheiro `Prog.java` que use a consola, assumindo que o diretório corrente é o de trabalho, usar o comando: **javac -cp ConsolePG.jar Prog.java**

Para executar a classe Prog depois de compilada, usar o comando: **java -cp .;ConsolePg.jar Prog**

Para usar a consola num projeto **IntelliJ** é necessário acrescentar à estrutura do projeto uma livreria constituída pelo ficheiro jar, seleccionando no menu: **File > Project Structure... > Libraries > "+" > Java** e seleccionar o `ConsolePG.jar` previamente copiado.

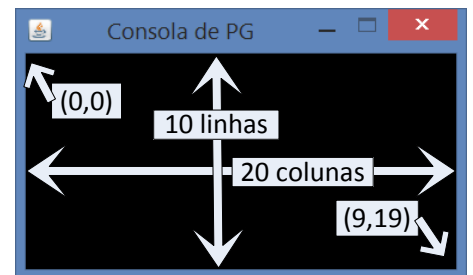
Para usar a consola num projeto **eclipse** é necessário acrescentar como um Jar externo ao conjunto de livrerias que consta no **Build Path** nas propriedades do projeto, seleccionando o projeto e seguindo no menu: **Project > Properties > Java Build Path > Libraries > Add External JARs...**

A janela da consola é aberta chamando o método `open()` indicando o título da janela, o número de linhas e de colunas.

Por exemplo, a chamada seguinte abre uma janela como a da figura. `Console.open("Consola de Pg",10,20);`

O canto superior esquerdo fica posicionado na linha 0 e na coluna 0. O canto inferior direito está na linha 9 e na coluna 19.

A janela não permite ser fechada nem redimensionada pelo utilizador.



Para escrever texto deve ser chamada uma das versões do método `print()` ou `println()` passando como parâmetro o conteúdo a escrever (char, String ou int). Por omissão, a primeira escrita é realizada a branco com fundo preto, na linha e na coluna 0.

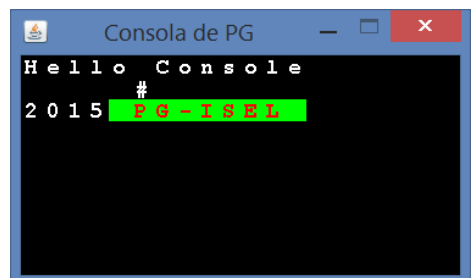
`Console.print("Hello Console");`

Se for chamado o método `cursor(int,int)` indicando a linha e a coluna, a próxima escrita será realizada nessa coordenada da consola, ficando o cursor posicionado no final da escrita ou, caso seja usado um dos métodos `println()` na coluna 0 da linha seguinte. O método `println()` ou a escrita explícita do carácter `'\n'` coloca o cursor na coluna 0 da linha seguinte.

`Console.cursor(1,5);`
`Console.println('#');`
`Console.print(2015);`

Se for chamado o método `color(int,int)` indicando a cor de escrita e de fundo, a escritas seguintes serão realizadas com essas cores.

`Console.color(Console.RED, Console.GREEN);`
`Console.print(" PG-ISEL ");`



As cores permitidas estão definidas como constantes do tipo `int` na classe `Console` com nomes sugestivos (`BLACK`, `WHITE`, `RED`, `GREEN`, `BLUE`, `YELLOW`, `MAGENTA`, `ORANGE`, `CYAN`, `PINK`, `BROWN`, `DARK_GRAY`, `GRAY`, `LIGHT_GRAY`), em que `BLACK=0`, `WHITE=1`, etc. A constante `Console.MAX_COLORS` indica o número total de cores disponíveis.

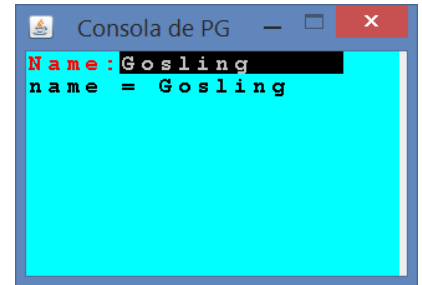
O método `clear()` apaga todo o conteúdo da janela escrevendo espaços com a cor de fundo corrente.

`Console.setBackground(Console.CYAN);`
`Console.clear();`
`Console.print("Name:");`

Classe Console usada em PG

Para ler texto introduzido pelo utilizador pode ser usado o método `nextLine(int)` em que se passa como parâmetro o comprimento máximo do texto a introduzir. Este método retorna uma referência para a `String` lida quando o utilizador premir <Enter>, ou `null` caso seja premido <Esc>. Durante a edição aparecerá um cursor e pode ser usado <Backspace> para apagar.

```
Console.color(Console.LIGHT_GRAY, Console.BLACK);
String name = Console.nextLine(12);
Console.println();
Console.color(Console.BLACK, Console.CYAN);
Console.print("name = "+name);
```



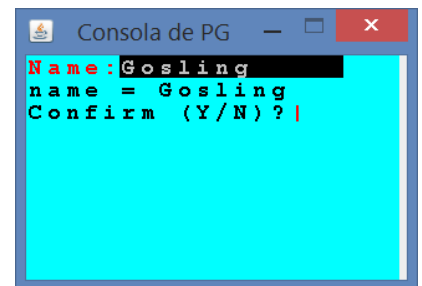
Para ler as teclas premidas pode ser usado o método `getChar()` ou `waitChar(int)`. O método `getChar()` retorna imediatamente a tecla que foi premida ou zero (`char` com código 0) se nenhuma tecla foi premida. O método `waitChar(int)` faz o mesmo que `getChar()`, mas caso nenhuma tecla tenha sido premida, espera até o máximo de tempo indicado em milissegundos que uma tecla seja premida antes de retornar zero.

Caso o tempo indicado no parâmetro de `waitChar(int)` seja zero, este só retornará quando for premida uma tecla.

O método `cursor(boolean)` permite tornar visível ou esconder o cursor.

Por omissão, quando é aberta a consola, o cursor não está visível.

```
Console.print("\nConfirm (Y/N)?");
Console.cursor(true);
char ans = Console.waitChar(10000); // 10seg
Console.cursor(false);
if (ans!=0 && (ans=='Y' || ans=='y')) Console.println("Ok");
```



O troço de código anterior fica até 10 segundos à espera que o utilizador prima uma tecla. Durante este tempo o cursor ficará visível logo a seguir ao '?'. Passados 10 segundos ou quando o utilizador premir uma tecla, o cursor é escondido e será escrito "Ok" caso seja premida a tecla 'Y'.

O método `getChar()` e o método `waitChar(int)` só permitem ler as teclas que são usadas na edição de texto (letras, dígitos, espaços, tab, enter, etc.). Para ler qualquer tecla, incluindo as de ação, por exemplo (Fn, Alt, Shift, setas de cursor, etc.), existem os métodos `getKeyPressed()` e `waitKeyPressed(int)` com o mesmo modo de funcionamento que `getChar()` e `waitChar()`, mas desta vez retornam um valor inteiro com o código da tecla ou -1 caso não seja premida qualquer tecla. Os códigos das teclas estão declarados como constantes começadas por `VK_` na classe `java.awt.event.KeyEvent`.

Se for necessário saber se qualquer tecla ou uma determinada tecla está premida no momento, devem ser usados os métodos `isKeyPressed()` ou `isKeyPressed(int)`. O primeiro retorna `true` caso esteja alguma tecla premida no momento, podendo saber qual foi se a seguir for chamado o método `getKeyPressed()`. O segundo retorna `true` se a tecla com o código indicado está premida no momento.

Para esperar que uma tecla seja libertada deve ser chamado o método `waitKeyReleased(int)`, passando como parâmetro o código da tecla.

Note-se que num determinado momento pode haver várias teclas premidas simultaneamente.

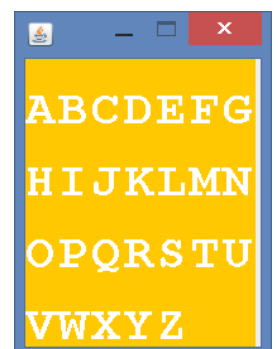
No final do programa, ou quando o programa pretender fechar a janela, deve ser chamado o método `close()`.
`Console.close();`

Antes de abrir a consola com o método `open()` pode-se especificar as cores de escrita e de fundo da consola, o tamanho da fonte de caracteres a usar e o fator de escala para a proporcionalidade das linhas e das colunas, com os métodos `color()`, `fontSize()` e `scaleFactor()`.

Depois da janela aberta já não é possível trocar de fonte nem o factor de escala.

Por omissão, a fonte tem tamanho 18 e o fator de escala é 1.0 para linhas e colunas.

```
Console.scaleFactor(2.0, 0.85);
Console.fontSize(38);
Console.color(Console.WHITE, Console.ORANGE);
Console.open("", 4, 7);
for(char c='A' ; c<='Z' ; ++c) Console.print(c);
```



Classe Console usada em PG

O seguinte programa exemplifica a utilização da classe console.

Este programa vai ocupando a consola com '#' consoante as teclas de cursor, usadas pelo utilizador.

Termina com a mensagem "Game over" quando o utilizador tentar ocupar uma posição já ocupada ou fora da consola e termina com a mensagem "Timeout" se não for premida uma tecla durante 5 segundos.


```
import isel.leic.pg.Console; // Permite usar Console sem indicar o package isel.leic.pg
import java.awt.event.KeyEvent; // Para usar as constantes VK_??? com o código de teclss

public class Snake {
    private static final int LINES=10, COLS=20;
    static int lin=LINES/2, col=COLS/2; // Posição corrente

    private static void writeMsg(String msg) {
        Console.cursor(LINES/2, (COLS-msg.length())/2);
        Console.color(WHITE, DARK_GRAY);
        Console.print(msg);
        Console.waitChar(3000);
    }

    public static void main(String[] args) {
        boolean[] path = new boolean[LINES*COLS];
        int key;
        boolean move;
        Console.open("Snake", LINES, COLS);
        Console.color(BLACK, RED);
        for(;;) {
            Console.cursor(lin,col);
            Console.print("#");
            path[lin*COLS+col]=true;
            if ((key= Console.waitKeyPressed(5000))==-1) {
                writeMsg("Timeout"); break;
            }
            move = true;
            switch (key) {
                case KeyEvent.VK_UP: if (lin>0) --lin; break;
                case KeyEvent.VK_DOWN: if (lin<LINES-1) ++lin; break;
                case KeyEvent.VK_LEFT: if (col>0) --col; break;
                case KeyEvent.VK_RIGHT: if (col<COLS-1) ++col; break;
                default: move = false;
            }
            if (move && path[lin*COLS+col]) {
                writeMsg("Game over"); break;
            }
            Console.waitKeyReleased(key);
        }
        Console.close();
    }
}
```



O método `exit(boolean)`, liga ou desliga a possibilidade de fechar automaticamente a consola premindo  da janela, conforme o parâmetro for `true` ou `false`, respetivamente. Inicialmente esta possibilidade está desligada.

ATENÇÃO: Quando a consola é fechada desta forma, o programa termina abruptamente sem executar as instruções desde o ponto corrente até ao final do método `main()`.

É possível detetar cliques na área da consola, ativando esta possibilidade com o método `mouseClick(boolean)` passando `true` como parâmetro. Passando `false` como parâmetro desativa-se novamente a deteção de cliques.

O método `getMouseClick()` retorna uma referência para um objeto da classe `isel.leic.pg.Location` que contém os campos públicos `lin` e `col` com as coordenadas da consola onde ocorreu o clique. O método retorna `false` se não existiu qualquer clique ou a deteção de cliques estiver desativada.

Quando a deteção de cliques está ligada, os métodos `getKeyPressed()` e `waitKeyPressed(int)` podem retornar o valor definido na constante `Console.MOUSE_CLICK` (-2) em vez de `Console.NO_KEY` (-1) para assinalar que não existe tecla premida mas foi detetado um clique.

Classe Console usada em PG

O seguinte programa exemplifica a utilização da console usando a detecção de cliques.

Cada vez que é dado um clique é colocado um bloco da cor selecionada (marcada com '*') na posição detetada. A barra de blocos que consta na primeira linha permite selecionar a próxima cor dos blocos a colocar. A cor pode ser selecionada andando para a esquerda ou para a direita com as teclas de cursor ou através de cliques na cor a selecionar. O programa termina quando for premida a tecla <Esc>.

```
import static java.awt.event.KeyEvent.*; // Usar diretamente os membros static de KeyEvent
import static isel.leic.pg.Console.*; // Usar diretamente os membros static de Console
import isel.leic.pg.Location; // Tipo retornado por getMouseClicked()

public class Paint {
    private static final int LINES = 20;
    private static final int COLS = 20;
    private static int currColor; // Cor selecionada

    public static void main(String[] args) {
        open("Paint", LINES + 1, COLS);
        mouseClicked(true);
        updatePallette(GREEN);
        draw();
        close();
    }

    private static void draw() {
        int key;
        do {
            key = waitKeyPressed(0);
            if (key==MOUSE_CLICKED)
                processClick(getMouseClicked());
            else if (key>0) {
                processKey(key);
                waitKeyReleased(key);
            }
        } while(key!= VK_ESCAPE);
    }

    private static void processClick(Location l) {
        if (l.lin>0) {
            cursor(l.lin, l.col);
            setBackground(currColor);
            print(' ');
        } else
            if (l.col<MAX_COLORS) updatePallette(l.col);
    }

    private static void processKey(int key) {
        if (key==VK_RIGHT)
            updatePallette(currColor+1);
        else if (key==VK_LEFT)
            updatePallette(currColor-1);
    }

    private static void updatePallette(int color) {
        cursor(0,0);
        currColor = color % MAX_COLORS;
        for (int i = 0; i < MAX_COLORS; i++) {
            color(i==WHITE || i==YELLOW ? BLACK : WHITE, i);
            print(currColor==i?'*':' ');
        }
    }
}
```

