Jaguar - PHP Graphic Library Documentation

Release 1.0.0

Hyyan Abo Fakher

May 30, 2014

Contents

Jaguar is a PHP5.3 OOP graphic library was built on the Great GD Library using the best practices to build a stable and fast layer for image manipulation and drawing.

Something to mention

Jaguar was built on php gd extension and this extension is so famous with its *"limitations"* in the php community like memory and transparent problems for example.

Thats's what Jaguar is for , overcome those *"limitations"* and let the programmer focus on the his/her application logic instead of spending hours trying to solve image transparent problem for example and finally moving to photoshop to solve the problem.

CHAPTER 1

Installation

Installation via composer

Simply add a dependency on hyyan/jaguar to your project's composer.json file. Here is a minimal example of a composer.json file that just defines a development-time dependency on Jaguar 1.0:

```
{
        "require-dev": {
                "hyyan/jaguar": "1.0.*"
        }
}
```

For a system-wide installation via Composer, you can run:

```
composer global require 'hyyan/jaguar=1.0.*'
```

Note: Make sure you have *~/.composer/vendor/bin/* in your path If you choosed the system-wide installation.

Contribute

Your contributions are more than welcome !

Start by forking Jaguar repository, write your feature, fix bugs, and send a pull request. If you modify Jaguar API, please update the API documentation in the Jaguar Docs repository

Note: Jaguar uses ApiGen project as documentation generator . take a look at the ApiGen project

You can update API documentation by running :

```
$ git clone http://www.github.com/hyyan/jaguar-docs
$ cd jaguar-docs
$ apigen --config="./aguar-apigen.neon"
           --template-config="path/to/apigen/templates/bootstrap/config.neon"
```

Note: Jaguar follows the Symfony standards If you're a beginner, you will find some guidelines about code contributions at Symfony.

CHAPTER 3

API

Jaguar API is available here.

3.1 Geometry System

Jaguar uses a very simple geometry system to make things easier for Jaguar and for the user . and this gemotry system is composed of three basic objects which are :

- **Coordinate** Composed of two integer *(x,y)* where x and y could be negative numbers.
- **Dimention** Composed of two integer *(width,height)*
- **Box** Composed of two objects *(Dimention,Coordinate)* where dimension is a **Dimesnion Object** and *coordinate* is a **Coordinate Object**

The following example demonstrates the usage of these objets.

```php
<?php

use Jaguar\Coordinate,
        Jaguar\Dimension,
        Jaguar\Box;

$geometry = array(
        $c = new Coordinate(4, 3),
        $d = new Dimension(100, 100),
        $b = new Box($d, $c)
);


foreach ($geometry as $object) {
        echo sprintf("%s\n",(string) $object);
}
```

Output :

```
Jaguar\Coordinate[x=4,y=3]
Jaguar\Dimension[width=100,height=100]
Jaguar\Box[Jaguar\Dimension[width=100,height=100],Jaguar\Coordinate[x=4,y=3]]
```

3.2 Color Model

Jaguar supports one color model which is the *RGB* model becuase it is the only supported model in GD Library which Jaguar is built on it .

Beside the RGB model Jaguar supports the GD special colors which is probably you will never need to deal with them directly but they are used widely inside Jaguar to create new effects and patterns.

The following example demonstartes some of the RGBColor main methods

```php
<?php

use Jaguar\Color\RGBColor;

// transparent(50) red
$rgb = new RGBColor(255, 0, 0, 50);

// also transparent(50) red  but using hex format
$hex = RGBColor::fromHex('#f00', 50);

// also transparent(50) red  but using the value of allocated color
$value = RGBColor::fromValue($hex->getValue());


echo sprintf(
                "%s\n%s\n%s"
                , (string) $rgb
                , (string) $hex
                , (string) $value
);
```

Output :

```
Jaguar\Color\RGBColor(855572480)[r=255,g=0,b=0,alpha=50]
Jaguar\Color\RGBColor(855572480)[r=255,g=0,b=0,alpha=50]
Jaguar\Color\RGBColor(855572480)[r=255,g=0,b=0,alpha=50]
```

3.3 Canvas

Canvas is the core of Jaguar which makes applying filters , drawing shapes and loading images from files or strings possible.it the warpper for all kinds of supported formats in Jaguar which are :

- Jpeg : Support *jpeg* format
- Png : Support *png* format
- Gif : Support *gif* format
- Gd : Support *Gd2* format

Probably you will never need to deal directly with any of those formats but you will only deal with Canvas object which is smart enough to guess the right format for a given image file or image string . But if you want to declare the formate by your own you can do it is straightforward

3.3.1 Creating New Canvas

The folowing example demonstrates creating a png canvas (w=100,h=100) filled with red color

```php
<?php

use Jaguar\Canvas,
        Jaguar\Dimension,
        Jaguar\Color\RGBColor;

// We want png canvas (by default png is the default format )
$canvas = new Canvas(
        new Dimension(100, 100)
        , Canvas::Format_PNG
);

$canvas->fill(new RGBColor(255))
           // save the canvas to the given path
           ->save('/save/path')
   // send the canvas to browser
           ->show();
```

Output :

3.3.2 Loading Canvas From Files Or Strings

As you can see Jaguar's canvas can create new image with no effort and also it can load images from files and string easily.

The following example will demonstrate loading images from a file.

```php
<?php

use Jaguar\Canvas;

// No need to define the format
// canvas will make a guess for you
// and find the right format
$canvas = new Canvas('images/canvas-1.jpg');

$canvas->save('/save/canvas-1.jpg')
                ->show();
```

Output :

3.3.3 Adding New Foramts

Canvas object is not limited to the current formats and it can be extended to support more by creating a new canvas which implements CanvasInterface or exteds the AbstractCanvas and a new format which implements the CanvasFactory

The following example demonstrates createing a new format called *Foo*

```php
<?php

use Jaguar\AbstractCanvas,
        Jaguar\CanvasFactory,
        Jaguar\Canvas;

/**
 * Foo Format
 */
class Foo extends AbstractCanvas
{

        protected function doLoadFromFile($file)
        {
                //put your code here
        }

        protected function doSave($path)
        {
                //put your code here
        }

        protected function getToStringProperties()
        {
                return array();
        }

}

/**
```

```
 * Foo Format Factory
 */
class FooFactory implements CanvasFactory
{

        public function getCanvas()
        {
                return new Foo();
        }

        public function getExtension($includeDot = true)
        {
                return ($includeDot) ? '.foo' : 'foo';
        }

        public function getMimeType()
        {
                return 'image/foo';
        }

        public function isSupported($file)
        {
                // check if the Foo format can handle
                // the given file
        }

}

/* add my new format to the canvas */
$canvas = new Canvas();
$canvas->addFactory('Foo', new FooFactory());

print_r($canvas->getFactories());
```

Output :

```
Array
(
        [factory.jpeg] => Jaguar\Factory\JpegFactory Object()
        [factory.gif] => Jaguar\Factory\GifFactory Object()
        [factory.png] => Jaguar\Factory\PngFactory Object()
        [factory.gd2] => Jaguar\Factory\GdFactory Object()
        [Foo] => FooFactory Object()

)
```

3.4 Drawables

Jaguar provides a fully-featured drawing API which helps you to draw almost everything from pixels to polygons using the simple DrawableInterface

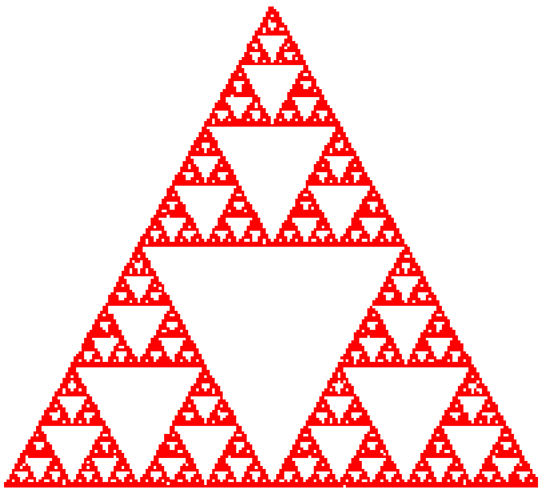
Drawable every object which implements the DrawableInterface so it can be drawn on a canvas. for example Rectangle is a drawables which implements the DrawableInterface

3.4.1 Drawing

The following example demonstrates a random drawing example from the php manual using Jaguar

```
<?php

use Jaguar\Drawable\Pixel,
        Jaguar\Color\RGBColor,
        Jaguar\Dimension,
        Jaguar\Coordinate,
        Jaguar\Canvas;

$w = 200;
$h = 200;

$canvas = new Canvas(new Dimension($w, $h));
$pixel = new Pixel();
$pixel->setColor(RGBColor::fromHex('#f00'));

$corners[0] = array('x' => 100, 'y' => 10);
$corners[1] = array('x' => 0, 'y' => 190);
$corners[2] = array('x' => 200, 'y' => 190);


for ($i = 0; $i < 100000; $i++) {

        $pixel->setCoordinate(new Coordinate(round($w), round($h)))
                        ->draw($canvas);

        $a = rand(0, 2);
        $w = ($w + $corners[$a]['x']) / 2;
        $h = ($h + $corners[$a]['y']) / 2;
}

$canvas->save('save/random-drawing.png')
                ->show();
```

Output :



3.4.2 Drawing With Styles

Drawing shapes and pixels using flat colors is good and nice but Jaguar supports more than that (thanks to GD Library) , it supports styles.

For now you can think of Jaguar styles as flat color replacement

Note: Not all drawables supports styles some drawables like Text for example can not be drawn with styles

See also:

Drawing texts with drawers

The following example will use the last example of random drawing but with the FillStyle this time to demonstrate the usage of styles.

```
<?php

use Jaguar\Drawable\Pixel,
        Jaguar\Drawable\Style\FillStyle,
        Jaguar\Dimension,
        Jaguar\Coordinate,
        Jaguar\Canvas;

$w = 200;
$h = 200;

$canvas = new Canvas(new Dimension($w, $h));
$style = new FillStyle(new Canvas('/images/canvas-1.jpg'));
$pixel = new Pixel();

$corners[0] = array('x' => 100, 'y' => 10);
$corners[1] = array('x' => 0, 'y' => 190);
$corners[2] = array('x' => 200, 'y' => 190);

for ($i = 0; $i < 100000; $i++) {

        $pixel->setCoordinate(new Coordinate(round($w), round($h)))
                        ->draw($canvas,$style);

        $a = rand(0, 2);
        $w = ($w + $corners[$a]['x']) / 2;
        $h = ($h + $corners[$a]['y']) / 2;
}

$canvas->save('images/random-darwing-with-style.png')
                ->show();
```

Output :

3.4.3 Texts

Texts is such a tough topic , actually it is not that easy as you might think and the Gd Library comes with basic supports for texts, But Jaguar provides a rich Text Object to help you overcome some limitations.

Although Text does not support styles but it supports text drawers. a text drawers can be attached to the text object to draw outlined text for example.

The following text drawers are supporte :

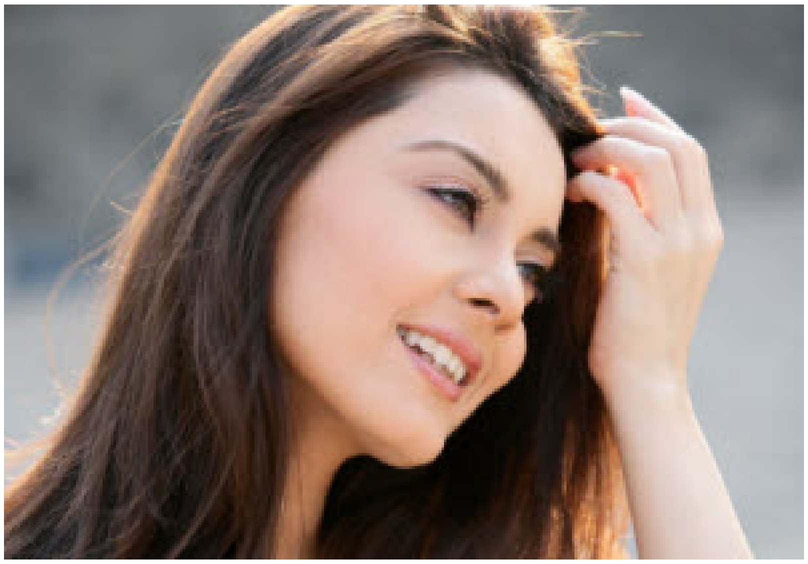


Figure 3.1: Draw Style

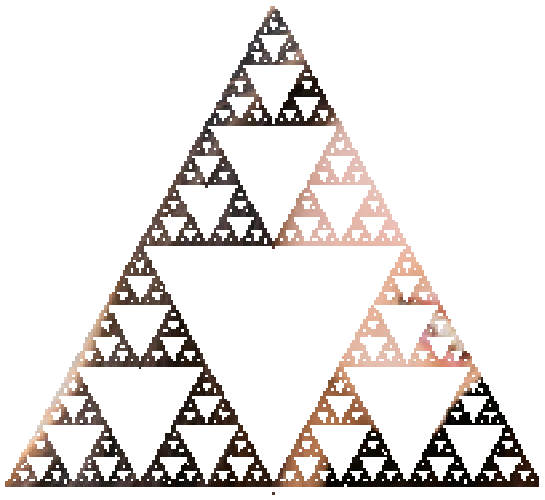


Figure 3.2: Drawing Result

- Shadow : draw text with shadow
- Outline : draw outlined text
- Plain : the default drawer

Text Drawers

The following example will draw text using both Shadow and Outline Drawers to demonstare the usage of Text Drawers

```php
<?php

use Jaguar\Canvas,
        Jaguar\Drawable\Text,
        Jaguar\Drawable\Text\Outline,
        Jaguar\Drawable\Text\Shadow,
        Jaguar\Coordinate,
        Jaguar\Font,
        Jaguar\Color\RGBColor;

$canvas = new Canvas('images/drawables-1.jpg');

// orange outline where outline width = 2
$outline = new Outline(RGBColor::fromHex('#f90'), 2);

// orange shadow where xoffset = yoffset = 1
$shadow = new Shadow(RGBColor::fromHex('#f90'), 1, 1);


// create white text to draw at (x=10,y=10)
$text = new Text(
                'Jaguar'
                , new Coordinate(10, 320)
                , RGBColor::fromHex('#fff')
);

// draw "jaguar" word using both drawer with impact font
$text->setFont(new Font('/fonts/impact.ttf', 20))
                ->setDrawer($outline)->draw($canvas)
                ->setDrawer($shadow)->draw($canvas)

                // draw "An Artist Touch" word in both drawers after translating
                // the text coordinate by "20" for x and "20" for y with animeace2_bld
                // font.

                ->setString('An Artist Touch')
                ->setCoordinate($text->getCoordinate()->translate(20, 30))
                ->setFont(new Font('/fonts/animeace2_bld.ttf', 10))
                ->setDrawer($outline)->draw($canvas)
                ->setDrawer($shadow)->draw($canvas);

$canvas->save('/images/text.jpg')->show();
```

Output :

Text Bounding Box

Sometimes you might need to measure the text width and height or you might just want to draw borders around the text or you might want ot create a simple cpatha project. so you need a tool to help achive that.

The following texts will demonstrate using text bound feature in Text object

```php
<?php

use Jaguar\Canvas,
        Jaguar\Drawable\Text,
        Jaguar\Coordinate,
        Jaguar\Drawable\Rectangle,
        Jaguar\Color\RGBColor;

$canvas = new Canvas('/images/drawables-2.jpg');

$text = new Text(
               'Jaguar Touch'
               , new Coordinate(250, 250)
               , RGBColor::fromHex('#fff')
);
$text->setFontSize(15)->draw($canvas);


// get bouding box
$bound = $text->getBoundingBox();

// draw text border
$rect = new Rectangle(
                $bound->getDimension()
                , $bound->getCoordinate()
                , RGBColor::fromHex('#fff')
);
$rect->draw($canvas);

$canvas->save('/images/text-bound.jpg')->show();
```

Output :

3.4.4 Borders

One of the main important features of the *Drawable* package is *Borders* which you can do wonders with them.

Borders in Jaguars supports both styles and drawers.for styles any Jaguar style can be applyed to the border and for drawers Jaguar support three drawers which are :

- **BorderIn** The default drawer, the border will be drawn over the canvas with no dimension changes in canvas
- **BorderFit** The canvas will be resized and the border will drawn around the new resized canvas but the result will be a canvas its dimension equals the orginal one.

- **BorderOut** Border will be drawn out of the canvas and that will lead to changes in the canvas dimesnsion

Draw Raw Borders

The following example will demonstrate how to draw border out using the border out drawer

```php
<?php

use Jaguar\Canvas,
        Jaguar\Drawable\Border,
        Jaguar\Drawable\Border\BorderOut,
        Jaguar\Color\RGBColor;

$canvas = new Canvas('/images/drawables-3.jpg');

$border = new Border(
                5 // border size
                , new RGBColor(41, 41, 41)
);

// draw border out
$border->setDrawer(new BorderOut())->draw($canvas);
$canvas->save('/images/border-out.jpg');
```

Output :

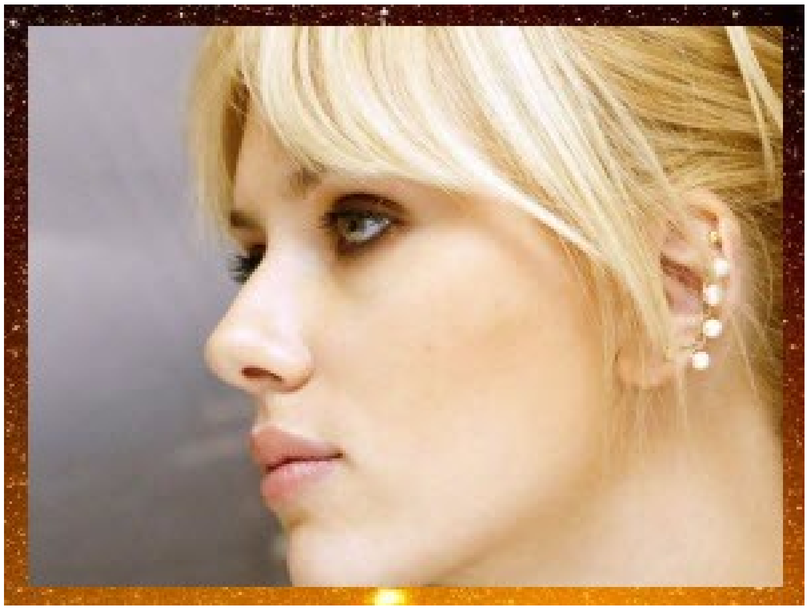
Draw Styled Border

The following example demonstrates drawing styled border using the default drawer (border in).

```php
<?php

use Jaguar\Canvas,
        Jaguar\Drawable\Border,
        Jaguar\Drawable\Style\FillStyle;
```

```
$canvas = new Canvas('/images/drawables-4.jpg');

$border = new Border(15);
$border->draw(
                $canvas
                , new FillStyle(new Canvas('/images/style.jpg'))
);

$canvas->save('/images/border-in-styled.jpg');
```

Output :



3.5 Gradients

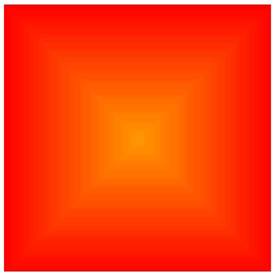
Let's face it , gradient is a necessary feature in any graphic library , it is important to generate styles and some kind of advanced image effects like Reflection for example. for that Jaguar provides a simple gradient generator which supports four kind of gradients which are :

- **CircleGradient** Same As Radial Gradient
- **LinearGradient** Supports both vertical and horizontal gradients
- RectangleGradient
- DiamondGradient

The following example deomnstrates generating the four kinds of gradients.

```php
<?php

use Jaguar\Canvas,
        Jaguar\Color\RGBColor,
        Jaguar\Dimension,
        Jaguar\Gradient\LinearGradient,
        Jaguar\Gradient\RectangleGradient,
        Jaguar\Gradient\CircleGradient,
        Jaguar\Gradient\DiamondGradient;

$canvas = new Canvas();

$start = RGBColor::fromHex('#000');
$end = RGBColor::fromHex('#fff');

$gradients = array(
        'linear-h' => new LinearGradient(LinearGradient::GRADIENT_HORIZONTAL, $start, $end),
        'linear-v' => new LinearGradient(LinearGradient::GRADIENT_VERTICAL, $start, $end),
        'diamond' => new DiamondGradient($start, $end),
        'rectangle' => new RectangleGradient($start, $end),
        'circle' => new CircleGradient($start, $end),
);


foreach ($gradients as $name => $gradient) {
        $canvas = $canvas->create(new Dimension(100, 100));
        $gradient->generate($canvas);
        $canvas->save(sprintf('images/%s.png', $name));
}
```

Output :



3.6 Actions

Most of the time actions that you can apply on a canvas is what really matters. For that Jagaur comes with a rich Actions Framework to make your life easier and to save your time. and all actions are super fast and non of them is pixel based.

3.6.1 Transformation

The most efficient way to learn actions is to use the Transformation Object which acts as Wraper for most used framework actions.

The following example will demonstrate how to to resize an image and watermark it using the transformation object

```php
<?php

use Jaguar\Canvas,
        Jaguar\Transformation,
        Jaguar\Coordinate,
        Jaguar\Dimension;

$actions = new Transformation(new Canvas('/images/actions-1.jpg'));
$actions->resize(new Dimension(400, 400))
                ->watermark(
                                new Canvas('/images/jaguar-logo.png')
                                , new Coordinate(20, 300)
                )
                ->getCanvas()
                ->save('/images/transformation-basic.jpg');
```

Output :

3.6.2 Box Action

Box action is one of the most interesting actions in Jaguar.this action can apply any kind of actions on selected area of a canvas

Note: Some libraries call this action selection

The following example demonstrates the usage of the box action

```php
<?php

use Jaguar\Canvas,
        Jaguar\Action\BoxAction,
        Jaguar\Action\Color\Grayscale,
        Jaguar\Box,
        Jaguar\Dimension,
        Jaguar\Coordinate,
        Jaguar\Transformation;

$canvas = new Canvas('/images/jaguar2.jpg');
$w = $canvas->getWidth();
$h = $canvas->getHeight();


$box1 = new Box(
                new Dimension($w / 2, $h / 2)
                , new Coordinate(0, 0)
);
$box2 = new Box(
                new Dimension($w / 2, $h / 2)
                , new Coordinate($w / 2, $h / 2)
);

$actions = new Transformation($canvas);
$actions->apply(new BoxAction(new Grayscale(), $box1))
                ->apply(new BoxAction(new Grayscale(), $box2))
                ->getCanvas()
                ->save('/images/box-action.png');
```

Output :

3.6.3 Layers Effect (Overlay)

Jaguar can simulate the layer effect by using the Overlay action

Overlay has the effect that black background pixels will remain black, white background pixels will remain white, but grey background pixels will take the colour of the foreground pixel.

Note: Jaguar comes prepared with some preset actions which use the overlay action heavily .

Most current preset actions are inspired from Marc Hibbins(http://marchibbins.com/dev/gd) but with different preset files

Current Preset Actions : , Canvas , Chrome , Dreamy , Monopin , Velvet , Vintage

The following example demonstrates creating vignette filter using overlay action with *vignette.gd2* preset which comes with Jaguar resources package

```php
<?php

use Jaguar\Canvas,
        Jaguar\Util,
        Jaguar\Transformation;

$transformation = new Transformation(new Canvas('/images/actions-1.jpg'));
$transformation->overlay(new Canvas(Util::getResourcePath('/Preset/vignette.gd2')))
               ->getCanvas()
               ->save('/images/overlay-action.jpg');
```

Output :

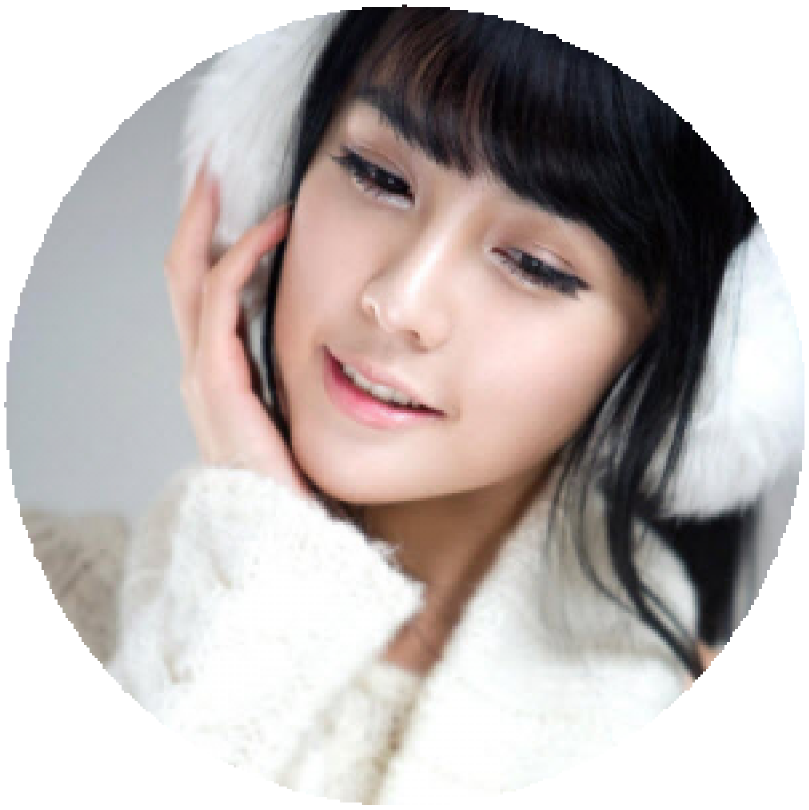
3.6.4 Creating New Actions

Creating new actions is so easy too , all you need to do is to create a new class which implements the ActionInterface or extends AbstractAction.

The following example demonstrates creating RoundedCanvas Action

```
<?php

use Jaguar\CanvasInterface,
        Jaguar\Drawable\Arc,
        Jaguar\Drawable\Style\FillStyle,
        Jaguar\Canvas,
        Jaguar\Dimension,
        Jaguar\Coordinate,
        Jaguar\Transformation,
        Jaguar\Action\AbstractAction;

class CirledCanvas extends AbstractAction
{

        /**
         * Apply circled canvas action
         *
         * @param \Jaguar\CanvasInterface $canvas
         */
        protected function doApply(CanvasInterface $canvas)
        {
                $dimension = $canvas->getDimension();
                $new = new Canvas($dimension);

                $arc = new Arc($dimension, new Coordinate(
                                $dimension->getWidth() / 2
                                , $dimension->getHeight() / 2
                ));
                $arc->fill(true)->draw($new, new FillStyle($canvas));

                $canvas->setHandler($new->getHandler());
        }

}

$canvas = new Canvas(file_get_contents('/images/jaguar2.jpg'));

$transformation = new Transformation($canvas);
$transformation->resize(new Dimension(300, 300))
                ->apply(new CirledCanvas())
                ->getCanvas()
                ->save('/images/circled-canvas.png');
```

Output :



3.6.5 Edge Detections

Jaguar comes prepared with wide range of algorithms for Edge Detections

Supported edge detections algorithms :

- GRADIENT_NORTH
- GRADIENT_WEST
- GRADIENT_EAST
- GRADIENT_NORTH_EAST
- GRADIENT_NORTH_WEST
- GRADIENT_SOUTH_EAST
- GRADIENT_SOUTH_WEST
- LINE_HORIZONTAL
- LINE_VERTICAL
- LINE_LEFT_DIAGONAL
- LINE_RIGHT_DIAGONAL
- SOBEL_HORIZONTAL
- SOBEL_VERTICAL
- PREWITT_HORIZONTAL
- PREWITT_VERTICAL
- SCHARR_HORIZONTAL
- SCHARR_VERTICAL
- EMBOSS_NORTH
- EMBOSS_NORTH_EAST
- EMOBOSS_EAST
- EMBOSS_SOUTH_EAST
- EMBOSS_SOUTH

- EMBOSS_SOUTH_WEST
- EMBOSS_WEST
- EMBOSS_NORTH_WEST
- LAPLACIAN_FILTER1
- LAPLACIAN_FILTER2
- LAPLACIAN_FILTER3
- LAPLACIAN_FILTER4
- FINDEDGE

The following example will dimonstrate how to use create Pencil Action using the *LAPLACIAN_FILTER3* edge detection algorithm

```
<?php

use Jaguar\Canvas,
        Jaguar\CanvasInterface,
        Jaguar\Action\AbstractAction,
        Jaguar\Action\EdgeDetection,
        Jaguar\Action\Color\Negate,
        Jaguar\Action\Color\Grayscale,
        Jaguar\Transformation;

class PencilAction extends AbstractAction
{

        /**
         * Apply Pencil Action
         *
         * @param CanvasInterface $canvas
         */
        protected function doApply(CanvasInterface $canvas)
        {
                $transformation = new Transformation($canvas);
                $transformation
                                ->apply(new Grayscale())
                                ->apply(new EdgeDetection(EdgeDetection::LAPLACIAN_FILTER3))
                                ->apply(new Negate());
        }

}

$transformation = new Transformation(new Canvas('/images\actions-1.jpg'));
$transformation->apply(new PencilAction())
                ->getCanvas()
                ->save('/images\pencil-action.jpg')
                ->show();
```

Output :

CHAPTER 4

Indices and tables

- *genindex*
- *search*