Изображение в Java - это объект класса Image или его потомка BufferedImage.

Класс Image

Абстрактный класс Image - это суперкласс для всех классов, представляющих графические изображения. Изображение должно быть получено способом, зависящим от платформы.

Поля класса Image

Модификатор и тип	Поле и описание
protected float	**accelerationPriority** Приоритет для ускорения этого изображения.
static int	**SCALE_AREA_AVERAGING** Используется алгоритм масштабирования изображения с усреднением по площади.
static int	**SCALE_DEFAULT** Использование алгоритма масштабирования изображения по умолчанию.
static int	**SCALE_FAST** Выбор алгоритма масштабирования изображения, который придает более высокий приоритет скорости масштабирования, чем плавности получаемого масштабируемого изображения.
static int	**SCALE_REPLICATE** Использование алгоритма масштабирования изображения, реализованного в классе `ReplicateScaleFilter`.
static int	**SCALE_SMOOTH** Выбор алгоритма масштабирования изображения, который отдает более высокий приоритет плавности изображения, нежели скорости масштабирования.
static Object	**UndefinedProperty** Объект `UndefinedProperty` должен быть возвращен всякий раз, когда извлекается свойство, которое не было определено для конкретного изображения.

Методы класса Image

Модификатор и тип	Метод и описание
`void`	**`flush`**`()` Сбрасывает все восстанавливаемые ресурсы, используемые этим объектом изображения. Сюда входят любые пиксельные данные, которые кэшируются для отображения на экране, а также любые системные ресурсы, которые используются для хранения данных или пикселей для изображения, если их можно воссоздать. Изображение сбрасывается в состояние, аналогичное тому, в котором оно было впервые создано, так что, если оно снова будет отображено, данные изображения должны быть воссозданы или извлечены снова из его источника. Примеры того, как этот метод влияет на определенные типы объектов изображения: • Объекты BufferedImage оставляют первичный растр, в котором хранятся их пиксели, нетронутыми, но удаляют всю кэшированную информацию об этих пикселях, такую как копии, загруженные на аппаратное обеспечение дисплея, для ускорения блицев. • Объекты изображения, созданные с помощью компонентных методов, которые принимают ширину и высоту, оставляют свой основной буфер пикселей нетронутым, но вся кэшированная информация освобождается так же, как это делается для объектов BufferedImage. • Объекты Volatile Image освобождают все свои пиксельные ресурсы, включая их первичную копию, которая обычно хранится на аппаратном обеспечении дисплея, где ресурсов недостаточно. Эти объекты могут быть позже восстановлены с помощью их метода `validate` • Объекты изображения, созданные инструментарием и классами компонентов, которые загружаются из файлов, URL-адресов или создаются `ImageProducer`, выгружаются, и все локальные ресурсы освобождаются. Эти объекты могут быть позже перезагружены из их исходного источника по мере необходимости при их визуализации, как и при их первом создании.
`float`	**`getAccelerationPriority`**`()` Возвращает текущее значение приоритета ускорения. **Возвращает:** `значение от 0 до 1 включительно, которое представляет текущее значение приоритета`

ImageCapabilities	**getCapabilities**(**GraphicsConfiguration** gc) Возвращает объект ImageCapabilities, который можно запросить относительно возможностей этого изображения в указанной графической конфигурации. Это позволяет программистам узнать больше информации времени выполнения для конкретного объекта изображения, который они создали. Например, пользователь может создать BufferedImage, но в системе может не остаться видеопамяти для создания изображения такого размера в данной графической конфигурации, поэтому, хотя объект может быть ускоряемым в принципе, он не имеет такой возможности в этой графической конфигурации. **Параметры:** gc – объект GraphicsConfiguration. Значение null для этого параметра приведет к получению возможностей изображения по умолчанию GraphicsConfiguration. **Возвращает:** Объект ImageCapabilities, который содержит возможности Image в указанной графической конфигурации.
abstract **Graphics**	**getGraphics**() Возвращает графический контекст для рисования изображения.
abstract int	**getHeight**(**ImageObserver** observer) Возвращает высоту изображения. Если высота еще не известна, метод возвращает -1 и указанный объект ImageObserver уведомляется об этом. **Параметры:** observer - объект, ожидающий загрузки изображения. **Возвращает:** высота этого изображения или -1, если высота еще не известна.
abstract **Object**	**getProperty**(**String** name, **ImageObserver** observer) Получает свойство этого изображения по имени. Имена отдельных свойств определяются различными форматами изображений. Если свойство не определено для конкретного изображения, этот метод возвращает объект UndefinedProperty. Если свойства для этого изображения еще не известны, метод возвращает null, и объект ImageObserver уведомляется об этом. Имя свойства "comment" должно использоваться для хранения необязательного комментария, который может быть

	представлен приложению как описание изображения, его источника или его автора. **Параметры:** `name` - имя свойства. `observer` - объект, ожидающий загрузки этого изображения. **Возвращает:** значение именованного свойства.
Image	**getScaledInstance**(int width, int height, int hints) Создает масштабированную версию этого изображения. Возвращается новый `Image` объект, который будет содержать изображение в указанными `width` и `height`. Новый `Image` объект может загружаться асинхронно, даже если исходное изображение уже загружено полностью. Если `width` или `height` является отрицательным числом, то подставляется значение для сохранения соотношения сторон исходных размеров изображения. Если оба `width` и `height` являются отрицательными, то используются исходные размеры изображения. **Параметры:** `width` - ширина, до которой нужно масштабировать изображение. `height` - высота, на которую нужно масштабировать изображение. `hints` - флаги, указывающие тип алгоритма, который будет использоваться для повторной выборки изображения. **Возвращает:** масштабированную версию изображения.
abstract **ImageProducer**	**getSource**() Возвращает объект, который создает пиксели для изображения. Метод вызывается классами фильтрации изображений и методами, которые выполняют преобразование и масштабирование изображений. **Возвращает:** производителя изображения, который создает пиксели для этого изображения.
abstract int	**getWidth**(**ImageObserver** observer)

	Возвращает ширину изображения. Если ширина еще не известна, этот метод возвращает `-1` и указанный объект `ImageObserver` уведомляется об этом.
	Параметры:
	`observer - объект, ожидающий загрузки изображения.`
	Возвращает:
	`ширину изображения или -1, если ширина еще не известна.`

`void`	**`setAccelerationPriority`**`(float priority)`
	Устанавливает, насколько важно ускорение для этого изображения.
	Сведения о приоритете используются для сравнения с приоритетами других объектов изображения при определении того, как использовать ограниченные ресурсы ускорения, такие как видеопамять. Если возможно ускорить это изображение и если для обеспечения этого ускорения недостаточно ресурсов, но они могут быть высвобождены путем лишения ускорения какого-либо другого изображения с более низким приоритетом. Изображения, имеющие одинаковый приоритет, занимают ресурсы в порядке очереди FIFO.
	Параметры:
	`priority - значение от 0 до 1 включительно, где более высокие значения указывают на большую важность ускорения. Значение 0 означает, что это изображение никогда не должно ускоряться. Другие значения используются для определения приоритета ускорения относительно других изображений.`
	Исключение:
	`IllegalArgumentException - если priority меньше 0 или больше 1.`

Поскольку класс Image является абстрактным, то создать объект этого класса с помощью оператора new невозможно. Его можно получить с помощью метода getToolkit класса Component:

Image img = getToolkit.getImage(“C:\\Images\\im.gif”);

или используя статический метод getDefaultToolkit класса Toolkit:

Image img = Toolkit.getDefaultToolkit.getImage(“C:\\Images\\im.gif”);

Кроме этого класс Toolkit содержит несколько методов createImage, возвращающих ссылку на объект Image:

Модификатор и тип	**Метод и описание**
Image	**createImage**(byte[] imagedata) Создает изображение, которое декодирует изображение, хранящееся в указанном массиве байтов. Данные должны быть в формате GIF или JPEG. **Параметры:** imagedata - массив байтов, представляющий данные изображения в поддерживаемом формате изображения. **Возвращает:** изображение.
abstract **Image**	**createImage**(byte[] imagedata, int imageoffset, int imagelength) Создает изображение, хранящееся в указанном массиве байтов, с указанным смещением и длиной. Данные должны быть в формате GIF или JPEG. **Параметры:** imagedata - массив байтов, представляющий данные изображения в поддерживаемом формате изображения. imageoffset - смещение начала данных в массиве. imagelength - длина данных в массиве. **Возвращает:** изображение.
abstract **Image**	**createImage**(**ImageProducer** producer) Создает изображение с указанным производителем изображений. **Параметры:** producer - производитель изображений, который будет использоваться. **ВОЗВРАТ:** изображение с указанным производителем изображений.
abstract **Image**	**createImage**(**String** filename) Возвращает изображение, которое получает пиксельные данные из указанного файла. Возвращаемое изображение представляет собой новый объект, который не будет использоваться совместно с любым другим вызывающим этот метод или его вариантом getImage. Этот метод сначала проверяет, установлен ли диспетчер безопасности. Если это так, метод вызывает метод security

manager checkReadс указанным файлом, чтобы гарантировать, что создание образа разрешено.

Параметры:

filename - имя файла, содержащего пиксельные данные в распознанном формате файла.

Возвращает:

изображение, которое получает свои пиксельные данные из указанного файла.

Выбрасывает исключения:

SecurityException - если диспетчер безопасности существует и его метод checkRead не разрешает операцию.

abstract **Image** **createImage**(**URL** url)

Возвращает изображение, которое получает пиксельные данные из указанного URL. Возвращаемое изображение представляет собой новый объект, который не будет использоваться совместно с любым другим вызывающим этот метод или его вариантом getImage.

Этот метод сначала проверяет, установлен ли диспетчер безопасности. Если это так, метод вызывает метод checkPermission менеджера безопасности с разрешением url.openConnection().getPermission(), чтобы гарантировать, что создание образа разрешено. Для совместимости с менеджерами безопасности до версии 1.2, если доступ запрещен с помощью FilePermission или SocketPermission, метод выбрасывает SecurityException, если соответствующий метод SecurityManager.checkXXX в стиле 1.1 также отклоняет разрешение.

Параметры:

url - URL-адрес для использования при извлечении пиксельных данных.

Возвращает:

изображение, которое получает свои пиксельные данные из указанного URL.

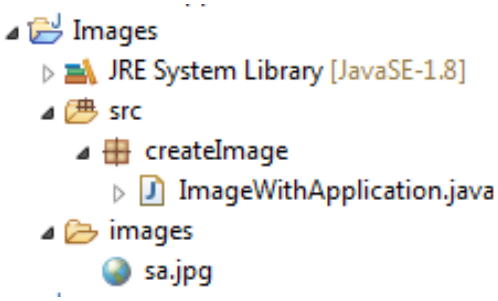
Выбрасывает исключения:

SecurityException - если менеджер безопасности существует и его метод checkPermission не разрешает операцию.

Пример создания изображения из файла

Перед выполнением примера разместите в каталоге images файл с именем sa.jpg.

Структура проекта




```
package createImage;
import java.awt.*;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class ImageWithApplication extends JFrame
{
      private static final long serialVersionUID = 1L;
      Image img2;
      int h, w;
public ImageWithApplication( )
  {
      super("Изображение в окне");
      //Получаем размеры экрана
    h = Toolkit.getDefaultToolkit().getScreenSize().height;
    w = Toolkit.getDefaultToolkit().getScreenSize().width;
    //Устанавливаем размеры окна равными размеру экрана
    setSize(w, h);
    //Отключаем менеджер компоновки
    this.setLayout(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }
 public void paint(Graphics g)
 {
       //Изображение из файла
       //Создаем панель
       JPanel p1 = new JPanel();
      //Определяем положение в окне и размер панели
       p1.setBounds(w/4, h/4, w/2, h/2);
       //Добавляем панель в окно
       this.add(p1);
       //Получаем графический контекст панели
       Graphics2D gp1 = (Graphics2D) p1.getGraphics();
       //Создаем изображение из файла
       Image img = Toolkit.getDefaultToolkit().getImage("images/sa.jpg");
       //Определяем размеры изображения
       int imh = img.getHeight(this);
       int imw = img.getWidth(this);
    //Масштабируем изображение так, чтобы оно уместилось на панели
       gp1.scale(0.5*(h/imh), 0.5*(w/imw));
       //Выводим изображение в панель
       gp1.drawImage(img, 0, 0, this);
  }
```

```
  public static void main( String args[])
  {
    new ImageWithApplication();
  }
}
```

Модель «Производитель - Потребитель»

AWT использует две модели обработки изображений; модель «Производитель - Потребитель» (Producer - Consumer) и модель с прямым доступом (immediate mode model).

Согласно модели «Производитель - Потребитель» производитель *генерирует сам или преобразует полученную из другого места* продукцию (в данном случае набор пикселей) и передает ее другим объектам (потребителям). Потребители *получают продукцию и при необходимости тоже преобразуют ее*. Только после этого создается объект класса Image и изображение выводится на экран. У одного производителя может быть несколько потребителей. Чтобы потребитель мог получать продукцию, он должен быть зарегистрирован у производителя. Производитель и потребитель взаимодействуют, вызывая методы друг друга.

Модель «Производитель – Потребитель» описана в двух интерфейсах пакета java.awt.image – ImageProducer и ImageConsumer.

Интерфейс ImageProducer

Интерфейс для объектов, которые могут создавать данные изображения для изображений. Каждое изображение содержит ImageProducer, который используется для восстановления изображения всякий раз, когда это необходимо, например, когда масштабируется новый размер изображения, или, когда запрашивается ширина или высота изображения.

Этот интерфейс реализуют классы MemoryImageSource, FilteredImageSource и RenderableImageProducer.

Методы интерфейса ImageProducer

Модификатор и тип	Способ и описание
void	**addConsumer**(**ImageConsumer** ic)

	Регистрирует ImageConsumerв ImageProducerдля доступа к данным изображения во время последующей реконструкции Image. ImageProducer по своему усмотрению может начать доставку данных изображения потребителю с помощью интерфейса ImageConsumer немедленно или когда следующая доступная реконструкция изображения запускается вызовом startProduction. **Параметры:** ic - указанный ImageConsumer
boolean	**isConsumer**(**ImageConsumer** ic) Определяет, зарегистрирован ли указанный ImageConsumerобъект в настоящее время в этом ImageProducer в качестве одного из его потребителей. **Параметры:** ic - указанный ImageConsumer **Возвращает:** true если указанный ImageConsumer зарегистрирован с этим ImageProducer; false в противном случае.
void	**removeConsumer**(**ImageConsumer** ic) Удаляет указанный ImageConsumer объект из списка потребителей, зарегистрированных в настоящее время для получения данных изображения. Удаление пользователя, который в данный момент не зарегистрирован, не считается ошибкой. ImageProducer следует прекратить отправку данных этому потребителю как можно скорее. **Параметры:** ic - указанный ImageConsumer

void	**requestTopDownLeftRightResend**(**ImageConsumer** ic)
	
	Запрашивает от имениImage Consumer, что ImageProducer попытался повторно отправить данные изображения еще раз в порядке СВЕРХУ ВНИЗ, чтобы можно было использовать алгоритмы преобразования более высокого качества, которые зависят от порядка приема пикселей, для получения лучшей выходной версии изображения.
	
	ImageProducer может игнорировать этот вызов, если он не может повторно отправить данные в таком порядке. Если данные могут быть повторно отправлены, ImageProducer должен ответить, выполнив следующий минимальный набор ImageConsumer вызовы методов:
	
	`ic.setHints(TOPDOWNLEFTRIGHT | < otherhints >);`
	
	`ic.setPixels(...);`
	
	`ic.imageComplete();`
	
	Параметры:
	ic - указанный ImageConsumer

void	**startProduction**(**ImageConsumer** ic) Регистрирует указанный объект ImageConsumer в качестве потребителя и запускает немедленную реконструкцию данных изображения, которые затем будут доставлены этому потребителю и любому другому потребителю, который, возможно, уже зарегистрирован у производителя. Этот метод отличается от метода addConsumer тем, что воспроизведение данных изображения должно быть запущено как можно скорее. **Параметры:** ic - указанный ImageConsumer

Класс MemoryImageSource

Этот класс является реализацией интерфейса ImageProducer, который использует массив для получения значений пикселей для изображения. Вот пример, который вычисляет изображение размером 100 x100 пикселей:

```
int w = 100;
int h = 100;
int pix[] = new int[w * h];
int index = 0;
for (int y = 0; y < h; y++) {
       int red = (y * 255) / (h - 1);
       for (int x = 0; x < w; x++) {
       int blue = (x * 255) / (w - 1);
      pix[index++] = (255 << 24) | (red << 16) | blue;
          }
       }
Image img = createImage(new MemoryImageSource(w, h, pix, 0, w));
```

MemoryImageSource способен управлять изображением в памяти, которое изменяется с течением времени, позволяя анимацию или пользовательский рендеринг. Ниже приводится пример, показывающий, как настроить источник анимации и сигнализировать об изменениях в данных:

```
int pixels[];
MemoryImageSource source;

public void init() {
int width = 50;
int height = 50;
```

```
int size = width * height;
pixels = new int[size];

int value = getBackground().getRGB();
for (int i = 0; i < size; i++) {
     pixels[i] = value;
          }
source = new MemoryImageSource(width, height, pixels, 0, width);
          source.setAnimated(true);
          image = createImage(source);
      }

public void run() {
Thread me = Thread.currentThread( );
me.setPriority(Thread.MIN_PRIORITY);

while (true) {
try {
Thread.sleep(10);
} catch( InterruptedException e ) {
return;
              }
// Изменение значений в массиве пикселей (x, y, w, h)
// Посылка новых данных заинтересованным потребителям
source.newPixels(x, y, w, h);
          }
      }
```

Класс FilteredImageSource

Конструктор:

```
public FilteredImageSource(ImageProducer orig,ImageFilter imgf)
```

Создает объект ImageProducer из существующего ImageProducer и объекта filter.

Параметры:

```
orig - определяет ImageProducer
imgf - определяет ImageFilter
```

Этот класс представляет собой реализацию интерфейса ImageProducer, который *берет существующее изображение и объект фильтра и использует их для создания данных изображения для новой отфильтрованной версии* исходного изображения. Следующий пример демонстрирует фильтрацию изображения, меняя местами красные и синие компоненты:

```
Image src = getImage("…");
ImageFilter colorfilter = new RedBlueSwapFilter();
Image img = createImage(new FilteredImageSource(src.getSource(),
```

```
                                                    colorfilter));
```

Класс ImageFilter будет описан далее.

Класс RenderableImageProducer

Класс адаптера, который реализует ImageProducer для обеспечения асинхронного создания отображаемого изображения. Размер ImageConsumer определяется масштабным коэффициентом преобразования usr2dev в RenderContext. Если значение RenderContext равно null, используется рендеринг RenderableImage по умолчанию. Этот класс реализует асинхронное производство, которое создает изображение в одном потоке с одним разрешением. Этот класс может быть подклассом для реализации версий, которые будут отображать изображение с использованием нескольких потоков. Эти потоки могут отображать либо одно и то же изображение с постепенно улучшающимся качеством, либо разные участки изображения с одним разрешением.

Конструктор:

```
RenderableImageProducer(RenderableImage rdblImage, RenderContext
                                                              rc)
```

Создает новый RenderableImageProducer из RenderableImage и RenderContext.

Параметры:

`rdblImage - визуализируемое изображение, которое должно быть отображено.`

`rc - RenderContext, используемый для создания пикселей.`

Модификатор и тип	Способ и описание
`void`	**`addConsumer`**`(`**`ImageConsumer`**` ic)` Добавляет ImageConsumer в список потребителей, заинтересованных в данных для этого изображения.
`boolean`	**`isConsumer`**`(`**`ImageConsumer`**` ic)` Определите, есть ли ImageConsumer в списке потребителей, которые в настоящее время заинтересованы в данных для этого изображения.
`void`	**`removeConsumer`**`(`**`ImageConsumer`**` ic)`

	Удалите ImageConsumer из списка потребителей, заинтересованных в данных для этого изображения.
void	**requestTopDownLeftRightResend**(**ImageConsumer** ic) Запрашивает, чтобы для данного ImageConsumer данные изображения были доставлены еще раз в порядке сверху вниз, слева направо.
void	**run**() Выполняемый метод для этого класса.
void	**setRenderContext**(**RenderContext** rc) Задает новый RenderContext для использования при следующем вызове startProduction().
void	**startProduction**(**ImageConsumer** ic) Добавляет ImageConsumer в список потребителей, заинтересованных в данных для этого изображения, и немедленно начинает доставку данных изображения через интерфейс ImageConsumer.

Пример использования MemoryImageSource

Вначале создается массив, содержащий цвет каждой точки изображения. Затем создается изображение, описанное этим массивом. Изображение выводится.

```
import java.awt.Graphics;
import java.awt.Image;
import java.awt.image.MemoryImageSource;

import javax.swing.JFrame;

public class InMemoryImage extends JFrame {
	private static final long serialVersionUID = 1L;
	//Размер создаваемого тзображения
	private int w =100;
	private int h =100;
	//Массив для создания изображения
	private int[] pix = new int[w*h];
	private Image img;

	InMemoryImage(String s) {
		super(s);

		//Запоняем массив для изображения
		int i = 0;
		for (int y = 0; y < h; y++) {
			int red = (y * 255) / (h - 1);
			for (int x = 0; x < w; x++) {
				int blue = (x * 255) / (w - 1);
				pix[i++] = (255 << 24) | (red << 16) | blue;
			}
```

```
            }
            this.setSize(200, 200);
            this.setVisible(true);
            this.setResizable(false);
            this.setDefaultCloseOperation(EXIT_ON_CLOSE);
      }

public void paint(Graphics g) {
      if (img == null)
                  img = this.createImage(new MemoryImageSource(w, h, pix, 0, w));
      g.drawImage(img, 50, 50, this);
}

      public static void main(String[] args) {
            new InMemoryImage("Изображение, созданное в памяти");

      }

      }
```

Интерфейс ImageConsumer

Интерфейс для объектов, проявляющих интерес к данным изображения через интерфейсы ImageProducer. Когда потребитель добавляется к производителю изображений, производитель предоставляет все данные об изображении, используя вызовы методов, определенные в этом интерфейсе.

Этот интерфейс реализуют следующие классы:

- AreaAveragingScaleFilter,
- BufferedImageFilter,
- CropImageFilter,
- GrayFilter,
- ImageFilter,
- PixelGrabber,
- ReplicateScaleFilter,
- RGBImageFilter.

Поля интерфейса ImageConsumer

Модификатор и тип	Поле и описание
static int	**COMPLETESCANLINES** Пиксели будут доставляться в виде (кратных) полных строк развертки одновременно.

static int	**IMAGEABORTED** Процесс создания изображения был намеренно прерван.
static int	**IMAGEERROR** При создании изображения была обнаружена ошибка.
static int	**RANDOMPIXELORDER** Пиксели будут доставлены в случайном порядке.
static int	**SINGLEFRAME** Изображение содержит одно статическое изображение.
static int	**SINGLEFRAMEDONE** Один кадр изображения завершен, но необходимо доставить еще несколько кадров.
static int	**SINGLEPASS** Пиксели будут доставлены за один проход.
static int	**STATICIMAGEDONE** Изображение завершено, и больше никаких пикселей или кадров для доставки не требуется.
static int	**TOPDOWNLEFTRIGHT** Пиксели будут доставляться в порядке сверху вниз, слева направо.

Методы интерфейса ImageConsumer

Модификатор и тип	Метод и описание
void	**imageComplete**(int status) Метод imageComplete вызывается, когда ImageProducer завершает доставку всех пикселей, содержащихся в исходном изображении, или когда завершен один кадр многокадровой анимации, или когда произошла ошибка при загрузке или создании изображения. ImageConsumer должен удалить себя из списка потребителей, зарегистрированных у производителя изображений, если он не заинтересован в получении последовательности кадров.
void	**setColorModel**(**ColorModel** model) Задает объект ColorModel, используемый для пикселей, о которых сообщается с помощью вызовов метода SetPixels. Обратите внимание, что каждый набор пикселей, доставляемых с помощью setPixels, содержит свой собственный объект ColorModel, поэтому не следует предполагать, что эта модель будет единственной, используемой для доставки значений пикселей. Примечательным

	случаем, когда можно увидеть несколько объектов цветовой модели, является отфильтрованное изображение, когда для каждого набора пикселей, которые он фильтрует, фильтр определяет, могут ли пиксели быть отправлены нетронутыми, используя исходную цветовую модель, или пиксели должны быть изменены (отфильтрованы) и переданы с использованием цветовой модели, более удобной для процессf фильтрации. **Параметры:** `model - указанный ColorModel`

`void`	**`setDimensions`**`(int width, int height)`
	Размеры исходного изображения сообщаются с помощью вызова метода setDimensions.
	Параметры:
	`width - ширина исходного изображения`
	`height - высота исходного изображения`

`void`	**`setHints`**`(int hintflags)` Задает подсказки, которые ImageConsumer использует для обработки пикселей, доставляемых ImageProducer. ImageProducer может доставлять пиксели в любом порядке, но ImageConsumer может масштабировать или преобразовывать пиксели в целевую цветовую модель более эффективно или с более высоким качеством, если он заранее знает некоторую информацию о том, как будут доставляться пиксели. Метод setHints должен вызываться перед любыми вызовами любого из методов setPixels с битовой маской подсказок о способе доставки пикселей. Если ImageProducer не следует рекомендациям для указанной подсказки, результаты не определены. **Параметры:** `hintflags - набор подсказок, которые ImageConsumer использует для обработки пикселей`

`void`	**`setPixels`**`(int x, int y, int w, int h,` **`ColorModel`** `model, byte[] pixels, int off, int scansize)`
	Доставляет пиксели изображения с помощью одного или нескольких вызовов этого метода.
	Каждый вызов определяет местоположение и размер прямоугольника исходных пикселей, которые содержатся в массиве пикселей. Указанный объект ColorModel должен использоваться для преобразования пикселей в соответствующие им цветовые и альфа-компоненты. Пиксель (m, n) хранится в массиве пикселей с индексом (n * scansize + m + off). Все пиксели, доставленные с помощью этого метода, сохраняются в виде **байтов**.
	Параметры:
	`x - координата X верхнего левого угла области пикселей, которую нужно установить`
	`y - координата Y верхнего левого угла области пикселей, которую нужно установить`
	`w - ширина области пикселей`
	`h - высота области пикселей`

model - указанный ColorModel

pixels - массив пикселей

off - смещение в pixelsмассив

scansize - расстояние от одного ряда пикселей до следующего в массиве pixels

void	**setPixels**(int x, int y, int w, int h, **ColorModel** model, int[] pixels, int off, int scansize) Пиксели изображения доставляются с помощью одного или нескольких вызовов метода SetPixels . Пиксели изображения доставляются с помощью одного или нескольких вызовов метода SetPixels. Каждый вызов определяет местоположение и размер прямоугольника исходных пикселей, которые содержатся в массиве пикселей. Указанный объект ColorModel следует использовать для преобразования пикселей в соответствующие им цветовые и альфа-компоненты. Пиксель (m, n) хранится в массиве пикселей с индексом (n * scansize + m + off). Все пиксели, доставленные с помощью этого метода, сохраняются как **целые числа**. все эти методы хранятся как целые числа. **Параметры:** x - координата X верхнего левого угла области пикселей, которую нужно установить y - координата Y верхнего левого угла области пикселей, которую нужно установить w - ширина области пикселей h - высота области пикселей model - указанный ColorModel pixels - массив пикселей off - смещение в массив pixels scansize - расстояние от одного ряда пикселей до следующего в массиве pixels
void	**setProperties**(**Hashtable**<?,?> props) Задает расширяемый список свойств, связанных с этим изображением. **Параметры:** props - список свойств, которые должны быть связаны с этим изображением

Моделирование цвета

Класс ColorModel

Абстрактный класс ColorModel *инкапсулирует методы для преобразования значения пикселя в цветовые компоненты* (например,

красный, зеленый и синий) и альфа-компонент. Чтобы отобразить изображение на экране, принтере или другом изображении, значения пикселей должны быть преобразованы в цветовые и альфа-компоненты. В качестве аргументов или возвращаемых значений из методов этого класса пиксели представлены в виде 32-битных целых чисел или в виде массивов примитивных типов. *Количество, порядок и интерпретация цветовых компонентов для ColorModel определяются его цветовым пространством ColorSpace*. ColorModel, используемый с данными пикселей, которые не содержат альфа-информации, рассматривает все пиксели как непрозрачные, что является альфа-значением 1.0.

Класс ColorModel поддерживает два представления значений пикселей. Значение пикселя может быть одним 32-разрядным int или массивом примитивных типов. API-интерфейсы Java для платформ 1.0 и 1.1 представляли пиксели как отдельные byte или единичные int значения. Для целей класса ColorModel аргументы значений пикселей были переданы как целые числа. API платформы Java 2 представил дополнительные классы для представления изображений. С объектами BufferedImageRenderedImage или, основанными на RasterSampleModel классах и, значения пикселей могут быть неудобны для представления в виде одного int. Теперь ColorModel имеет методы, которые принимают значения пикселей, представленные в виде массивов примитивных типов. Примитивный тип, используемый конкретным объектом ColorModel, называется его типом передачи.

Объекты ColorModel , используемые с изображениями, для которых значения пикселей неудобно представлять в виде одного int, выдают IllegalArgumentExceptionпри вызове методов, принимающих один аргумент int pixel. Подклассы ColorModel должны указывать условия, при которых это происходит. Это не происходит с объектами DirectColorModel или IndexColorModel.

В настоящее время типами передачи, поддерживаемыми Java 2D API, являются DataBuffer.TYPE_BYTE, DataBuffer.TYPE_USHORT,

DataBuffer.TYPE_INT, DataBuffer.TYPE_SHORT, DataBuffer.TYPE_FLOAT и DataBuffer.TYPE_DOUBLE. Большинство операций рендеринга будут выполняться намного быстрее при использовании цветовых моделей и изображений, основанных на первых трех из этих типов. Кроме того, некоторые операции фильтрации изображений не поддерживаются для цветовых моделей и изображений, основанных на последних трех типах. Тип переноса для конкретного объекта ColorModel указывается при создании объекта либо явно, либо по умолчанию. Все подклассы ColorModel должны указывать, каковы возможные типы передачи и как определяется количество элементов в примитивных массивах, представляющих пиксели.

Для BufferedImages, тип передачи этого Raster и Raster-объекта SampleModel (доступен с помощью методов getTransferType этих классов) должен совпадать с ColorModel типом передачи. Количество элементов в массиве, представляющем пиксель для Raster и SampleModel (доступное из методов getNumDataElements этих классов), должно совпадать с количеством элементов в ColorModel.

Алгоритм, используемый для преобразования значений пикселей в цветовые и альфа-компоненты, зависит от подкласса. Например, не обязательно существует взаимно однозначное соответствие между образцами, полученными из компонентов SampleModel объекта BufferedImage Raster и цвета / альфа. Даже при наличии такого соответствия количество битов в выборке не обязательно совпадает с количеством битов в соответствующем цветовом / альфа-компоненте. Каждый подкласс должен указывать, как выполняется преобразование значений пикселей в цветовые / альфа-компоненты.

Методы в классе ColorModel используют два разных представления цветовых и альфа-компонентов - нормализованную форму и ненормализованную форму. В нормализованной форме каждый компонент представляет собой значение float между некоторыми минимальными и максимальными значениями. Для альфа-компонента минимальное значение

равно 0.0, а максимальное - 1.0. Для цветовых компонентов минимальные и максимальные значения для каждого компонента могут быть получены из объекта ColorSpace. Эти значения часто будут равны 0.0 и 1.0 (например, нормализованные значения компонентов для цветового пространства sRGB по умолчанию варьируются от 0.0 до 1.0), но некоторые цветовые пространства имеют значения компонентов с разными верхними и нижними пределами. Эти ограничения могут быть получены с помощью методов getMinValue и getMaxValue класса ColorSpace. Нормализованные значения цветовых компонентов не умножаются предварительно. Все ColorModels должны поддерживать нормализованную форму.

В ненормализованной форме каждый компонент представляет собой целое значение без знака в диапазоне от 0 до $2^n - 1$, где n - количество значащих битов для конкретного компонента. Если значения пикселей для конкретного ColorModel представляют образцы цветов, предварительно умноженные на альфа-образец, ненормализованные значения цветовых компонентов также умножаются. Ненормализованная форма используется только с экземплярами ColorModel, ColorSpace минимальные значения компонентов которых равны 0.0 для всех компонентов и максимальные значения 1.0 для всех компонентов. Ненормализованная форма для цветовых и альфа-компонентов может быть удобным представлением, для ColorModel которого все значения нормализованных компонентов лежат в диапазоне от 0.0 до 1.0. В таких случаях целочисленное значение 0 соответствует 0.0, а значение $2^n - 1$ соответствует 1.0. В других случаях, например, когда нормализованные значения компонентов могут быть как отрицательными, так и положительными, ненормализованная форма неудобна. Такие объекты ColorModel генерируют IllegalArgumentException, когда вызываются методы, включающие ненормализованный аргумент. Подклассы ColorModel должны указывать условия, при которых это происходит.

Прямыми потомками абстрактного класса ColorModel являются классы ComponentColorModel, IndexColorModel, PackedColorModel.

Поля класса ColorModel

Модификатор и тип	Поле и описание
protected int	**pixel_bits** Общее количество битов в пикселе.
protected int	**transferType** Тип данных массива, используемого для представления значений пикселей.

Конструкторы класса ColorModel

Модификатор	Конструктор и описание
	ColorModel(int bits) Создает ColorModel, который преобразует пиксели с указанным количеством битов в цветовые / альфа-компоненты. Цветовым пространством по умолчанию является RGB ColorSpace, то есть sRGB. Предполагается, что значения пикселей содержат альфа-информацию. Если цветовая и альфа-информация представлены в значении пикселя в виде отдельных пространственных полос, предполагается, что цветовые полосы не будут предварительно умножены на альфа-значение. Тип прозрачности - java.awt.Transparency . TRANSLUCENT. Тип передачи будет наименьшим из DataBuffer.TYPE_BYTE DataBuffer.TYPE_USHORT, или DataBuffer.TYPE_INT, который может содержать один пиксель (или DataBuffer.TYPE_UNDEFINED, если бит больше 32). Поскольку этот конструктор не имеет информации о количестве битов на цвет и альфа-компонент, любой подкласс, вызывающий этот конструктор, должен переопределять любой метод, который требует этой информации. **Параметры:** bits - количество бит в пикселе **Выбрасывает:** IllegalArgumentException - если количество битов в bits меньше 1
protected	**ColorModel**(int pixel_bits, int[] bits, **ColorSpace** cspace, boolean hasAlpha, boolean isAlphaPremultiplied, int transparency, int transferType) Создает ColorModel, который преобразует значения пикселей в цветовые / альфа-компоненты. Цветовые компоненты будут в указанном ColorSpace. pixel_bits это количество битов в значениях пикселей. Массив битов задает количество значащих битов для каждого цвета и альфа-компонента. Его длина должна

быть равна количеству компонентов вColorSpace, если в значениях пикселей нет альфа-информации, или на один больше этого числа, если есть альфа-информация. hasAlpha указывает, присутствует ли альфа-информация или нет. boolean isAlphaPremultiplied определяет, как интерпретировать значения пикселей, в которых информация о цвете и альфа представлены в виде отдельных пространственных полос. Если boolean = true то, предполагается, что образцы цветов были умножены на альфа-образец. Transparency указывает, какие альфа-значения могут быть представлены этой цветовой моделью. Тип переноса - это тип примитивного массива, используемого для представления значений пикселей. Обратите внимание, что массив bits содержит количество значащих битов на цвет / альфа-компонент после преобразования из значений пикселей. Например, для IndexColorModel при pixel_bits значении, равном 16, массив bits может содержать четыре элемента с каждым элементом, равным 8.

Параметры:

pixel_bits - количество битов в значениях пикселей

bits - массив, который определяет количество значащих битов для каждого цвета и альфа-компонента

cspace - указанный ColorSpace

hasAlpha - trueесли присутствует альфа-информация; falseв противном случае

isAlphaPremultiplied – true если предполагается, что образцы цвета предварительно умножаются на альфа-образцы; false в противном случае

transparency - какие альфа-значения могут быть представлены этой цветовой моделью

transferType - тип массива, используемого для представления значений пикселей

Выдает:

IllegalArgumentException - если длина битового массива меньше, чем количество цветовых или альфа-компонентов в немColorModel, или если прозрачность не является допустимым значением.

IllegalArgumentException - если сумма количества битов в bitsменьше 1 или если какой-либо из элементов в bitsменьше 0.

Методы класса ColorModel

Модификатор и тип	Метод и описание
ColorModel	coerceData(WritableRaster raster, boolean isAlphaPremultiplied)

Принудительно приводит растровые данные к состоянию, указанному в переменной `isAlphaPremultiplied`, предполагающей, что данные в настоящее время правильно описаны этим `ColorModel`.

Он может умножать или делить данные цветного растра на альфа-значение или ничего не делать, если данные находятся в правильном состоянии. Если данные должны быть принудительно обработаны, этот метод также вернет экземпляр this `ColorModel` с соответствующим `isAlphaPremultiplied` установленным флагом. Этот метод выдаст a `UnsupportedOperationException`, если он не поддерживается этим `ColorModel`. Поскольку `ColorModel`это абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны переопределять этот метод, поскольку реализация в этом абстрактном классе выдает `UnsupportedOperationException`.

Параметры:

`raster - WritableRasterданные`

`isAlphaPremultiplied - trueесли альфа-значение предварительно умножается; falseв противном случае`

ВОЗВРАТ:

`ColorModelобъект, представляющий принудительно преобразованные данные.`

SampleModel **createCompatibleSampleModel**(int w, int h)

Создает `SampleModel` с указанной шириной и высотой, которая имеет макет данных, совместимый с этим `ColorModel`.

Поскольку `ColorModel` это абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны переопределять этот метод, поскольку реализация в этом абстрактном классе выдает `UnsupportedOperationException`.

Параметры:

`w - ширина, применяемая к новому SampleModel`

`h - высота, применяемая к новому SampleModel`

ВОЗВРАТ:

`SampleModelобъект с заданными шириной и высотой.`

Выдает:

`UnsupportedOperationException - если этот метод не поддерживается этим ColorModel`

WritableRaster **createCompatibleWritableRaster**(int w, int h)

Создает `WritableRaster` с указанной шириной и высотой, которая имеет макет данных (`SampleModel`) совместим с этим `ColorModel`.

Поскольку `ColorModel`это абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны

переопределять этот метод, поскольку реализация в этом абстрактном классе выдает `UnsupportedOperationException`.

Параметры:

`w - ширина, применяемая к новому WritableRaster`

`h - высота, применяемая к новому WritableRaster`

ВОЗВРАТ:

`WritableRasterобъект с заданными шириной и высотой.`

Выдает:

`UnsupportedOperationException - если этот метод не поддерживается этим ColorModel`

`boolean`	**`equals`**`(`**`Object`**` obj)` Проверяет, соответствует ли указанное `Object` является экземпляром `ColorModel` и если оно равно этому `ColorModel`.
`void`	**`finalize`**`()` Распоряжается системными ресурсами, связанными с этим `ColorModel` как только это `ColorModel` на него больше не ссылаются.

`abstract int`	**`getAlpha`**`(int pixel)`
	
	Возвращает альфа-компонент для указанного пикселя в масштабе от 0 до 255.
	
	Значение пикселя указывается как int . Выбрасывается`IllegalArgumentException`, если значения пикселей для этого `ColorModel`не могут быть удобно представлены в виде одного int .
	
	Параметры:
	`pixel - указанный пиксель`
	
	ВОЗВРАТ:
	`значение альфа-компонента указанного пикселя.`

`int`	**`getAlpha`**`(`**`Object`**` inData)` Возвращает альфа-компонент для указанного пикселя в масштабе от 0 до 255. Значение пикселя определяется массивом элементов данных типа transferType, переданных в качестве ссылки на объект. Если InData не является примитивным массивом типа transferType, `ClassCastException`выбрасывается а . Выбрасывается`ArrayIndexOutOfBoundsException`, если `inData`значение недостаточно велико, чтобы содержать значение пикселя для этого `ColorModel`. Если это `transferType`не поддерживается, `UnsupportedOperationException`будет выдан а . Поскольку `ColorModel`это абстрактный класс, любой экземпляр

должен быть экземпляром подкласса. Подклассы наследуют реализацию этого метода, и если они не переопределяют его, этот метод генерирует исключение, если подкласс использует `transferType`другой, чем `DataBuffer.TYPE_BYTE`, `DataBuffer.TYPE_USHORT`, или `DataBuffer.TYPE_INT`.

Параметры:

`inData` - указанный пиксель

ВОЗВРАТ:

альфа-компонент указанного пикселя, масштабируемый от 0 до 255.

Выдает:

`ClassCastException` - if `inData`не является примитивным массивом типа `transferType`

`ArrayIndexOutOfBoundsException` - если `inData`недостаточно велико, чтобы содержать значение пикселя для этого `ColorModel`

`UnsupportedOperationException` - если это `tranferType`не поддерживается этим `ColorModel`

`WritableRaster`	**`getAlphaRaster`**(**`WritableRaster`** raster)

Возвращает `Raster` представляющий альфа-канал изображения, извлеченного из входных данных `Raster`при условии, что значения пикселей этого `ColorModel` представлять цветовую и альфа-информацию в виде отдельных пространственных полос.

Этот метод предполагает, что `Raster` объекты, связанные с таким `ColorModel`, сохраняют альфа-полосу, если она присутствует, в качестве последней полосы данных изображения. Возвращает`null` , если с этим не связан отдельный пространственный альфа-канал `ColorModel`. Если это `IndexColorModel`объект, который имеет альфа-код в таблице поиска, этот метод вернет`null`, поскольку нет пространственно дискретного альфа-канала. Этот метод создаст новую `Raster` (но будет совместно использовать массив данных). Поскольку `ColorModel` это абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны переопределять этот метод, чтобы получить любое поведение, отличное от возврата`null` , поскольку реализация в этом абстрактном классе возвращает `null`.

Параметры:

`raster` - указанный `Raster`

ВОЗВРАТ:

aRaster, представляющий альфа-канал изображения, полученный из указанного `Raster`.

`abstract int`	**`getBlue`**(`int pixel`)

Возвращает компонент синего цвета для указанного пикселя, масштабируемый от 0 до 255 в цветовом пространстве RGB по умолчанию, sRGB.

При необходимости выполняется преобразование цвета. Значение пикселя указывается как int. Выбрасывается `IllegalArgumentException`, если значения пикселей для этого `ColorModel`не могут быть удобно представлены в виде одного int. Возвращаемое значение не является предварительно умноженным значением, например, если альфа предварительно умножается, этот метод делит его перед возвратом значения. Если альфа-значение равно 0, то синее значение равно 0.

Параметры:

`pixel - указанный пиксель`

ВОЗВРАТ:

`значение синего компонента указанного пикселя.`

`int` **`getBlue`**`(`**`Object`** `inData)`

Возвращает компонент синего цвета для указанного пикселя, масштабируемый от 0 до 255 в RGB по умолчанию `ColorSpace`, sRGB.

При необходимости выполняется преобразование цвета. Значение пикселя определяется массивом элементов данных типа transferType, переданных в качестве ссылки на объект. Возвращаемое значение не является предварительно умноженным значением. Например, если альфа-значение предварительно умножается, этот метод делит его, прежде чем возвращать значение. Если альфа-значение равно 0, то синее значение будет равно 0. Если `inData`это не примитивный массив типа transferType, `ClassCastException`то выбрасывается а. Ан`ArrayIndexOutOfBoundsException` выбрасывается , если `inData`он недостаточно велик , чтобы содержать значение пикселя для этого `ColorModel`. Если это `transferType`не поддерживается, `UnsupportedOperationException`будет выброшен символ а. Поскольку `ColorModel`это абстрактный класс, любой экземпляр должен быть экземпляром подкласса. Подклассы наследуют реализацию этого метода, и если они не переопределяют его, этот метод генерирует исключение, если подкласс использует `transferType`значение, отличное от `DataBuffer.TYPE_BYTE`, `DataBuffer.TYPE_USHORT`, или `DataBuffer.TYPE_INT`.

Параметры:

`inData - массив значений пикселей`

ВОЗВРАТ:

`значение синего компонента указанного пикселя.`

Выдает:

`ClassCastException - if inDataне является примитивным массивом типа transferType`

`ArrayIndexOutOfBoundsException - если inDataнедостаточно велико, чтобы содержать значение пикселя для этого ColorModel`

`UnsupportedOperationException - если это tranferTypeне подтверждается этим ColorModel`

ColorSpace	**getColorSpace**() Возвращает ColorSpace связанные с этим ColorModel.
int[]	**getComponents**(int pixel, int[] components, int offset) Возвращает массив ненормализованных цветовых / альфа-компонентов, заданных пикселем в этом ColorModel. Значение пикселя указывается как int. IllegalArgumentException будет выдан, если значения пикселей для этого ColorModel не удобно представлять в виде одного int или если значения цветовых компонентов для этого ColorModel не удобно представлять в ненормализованной форме. Например, этот метод можно использовать для извлечения компонентов для определенного значения пикселя в DirectColorModel. Если массив компонентов равен null , то будет выделен новый массив. Будет возвращен массив компонентов. Компоненты цвета / альфа хранятся в массиве компонентов, начиная с offset(даже если массив выделен этим методом). ArrayIndexOutOfBoundsExceptionВыбрасывается, если массив компонентов не nullявляется и не является достаточно большим, чтобы вместить все цветовые и альфа-компоненты (начиная со смещения). Поскольку ColorModelэто абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны переопределять этот метод, поскольку реализация в этом абстрактном классе выдает UnsupportedOperationException. **Параметры:** pixel - указанный пиксель components - массив для получения цветовых и альфа-компонентов указанного пикселя offset - смещение в componentsмассив, с которого начинается сохранение цветовых и альфа-компонентов **ВОЗВРАТ:** массив, содержащий цветовые и альфа-компоненты указанного пикселя, начиная с указанного смещения. **Выдает:** UnsupportedOperationException - если этот метод не поддерживается этим ColorModel
int[]	**getComponents**(**Object** pixel, int[] components, int offset) Возвращает массив ненормализованных цветовых / альфа-компонентов, заданных пикселем в этом ColorModel.

int[]	**getComponentSize**() Возвращает массив с количеством битов для каждого цветового / альфа-компонента. Массив содержит цветовые компоненты в порядке, указанном символом ColorSpace, за которым следует альфа-компонент, если он присутствует. **ВОЗВРАТ:** массив с количеством битов на цвет / альфа-компонент
int	**getComponentSize**(int componentIdx) Возвращает количество битов для указанного цветового / альфа-компонента. Цветовые компоненты индексируются в порядке, указанном ColorSpace. Как правило, этот порядок отражает название типа цветового пространства. Например, для TYPE_RGB индекс 0 соответствует красному, индекс 1 - зеленому, а индекс 2 - синему. Если это ColorModel поддерживает альфа, альфа-компонент соответствует индексу, следующему за последним цветовым компонентом. **Параметры:** componentIdx - индекс цветового / альфа-компонента **ВОЗВРАТ:** количество битов для цветового / альфа-компонента в указанном индексе. **Выбрасывает:** ArrayIndexOutOfBoundsException - если componentIdxбольше количества компонентов или меньше нуля NullPointerException - если количество битов массива равно null
int	**getDataElement**(float[] normComponents, int normOffset) Возвращает значение пикселя, представленное в виде int в этом ColorModel, учитывая массив нормализованных цветовых / альфа-компонентов. Этот метод выдастIllegalArgumentException, если значения пикселей для этого ColorModelне представимы как единое intцелое. ArrayIndexOutOfBoundsExceptionЕсли normComponentsмассив недостаточно велик, чтобы вместить все цветовые и альфа-компоненты (начиная с normOffset), создается значение An . Поскольку ColorModelэто абстрактный класс, любой экземпляр является экземпляром подкласса. Реализация этого метода по умолчанию в этом абстрактном классе сначала преобразует из нормализованной формы в ненормализованную форму, а затем вызывает getDataElement(int[], int). Подклассы, которые

	могут иметь экземпляры, которые не поддерживают ненормализованную форму, должны переопределять этот метод. **Параметры:** normComponents - массив нормализованных цветовых и альфа-компонентов normOffset - индекс, с normComponentsкоторого начинается извлечение цветовых и альфа-компонентов **ВОЗВРАТ:** значение intпикселя в этом ColorModelсоответствует указанным компонентам. **Выдает:** IllegalArgumentException - если значения пикселей для этого ColorModelне удобно представлять в виде одного int ArrayIndexOutOfBoundsException - если normComponentsмассив недостаточно велик для хранения всех цветовых и альфа-компонентов, начиная с normOffset
int	**getDataElement**(int[] components, int offset) Возвращает значение пикселя, представленное в виде int в этом ColorModel, учитывая массив ненормализованных цветовых / альфа-компонентов. Этот метод выдастIllegalArgumentException, если значения компонентов для этого ColorModelне удобно представлять в виде одного intили если значения цветовых компонентов для этого ColorModelне удобно представлять в ненормализованной форме. ArrayIndexOutOfBoundsExceptionВыбрасываетсяcomponents, если массив недостаточно велик, чтобы вместить все цветовые и альфа-компоненты (начиная с offset). Поскольку ColorModelэто абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны переопределять этот метод, поскольку реализация в этом абстрактном классе выдает UnsupportedOperationException. **Параметры:** components - массив ненормализованных цветовых и альфа-компонентов offset - индекс, с componentsкоторого следует начинать извлечение цветовых и альфа-компонентов **ВОЗВРАТ:** значение intпикселя в этом ColorModelсоответствует указанным компонентам. **Выдает:**

IllegalArgumentException - если значения пикселей для этого ColorModelне могут быть удобно представлены в виде одного int

IllegalArgumentException - если значения компонентов для этого ColorModelне могут быть удобно представлены в ненормализованной форме

ArrayIndexOutOfBoundsException - если componentsмассив недостаточно велик, чтобы вместить все цветовые и альфа-компоненты, начиная с offset

UnsupportedOperationException - если этот метод не поддерживается этим ColorModel

Object **getDataElements**(float[] normComponents, int normOffset, **Object** obj)

Возвращает массив элементов данных, представляющий пиксель в этом ColorModel, учитывая массив нормализованных цветовых / альфа-компонентов.

Затем этот массив может быть передан setDataElementsметоду WritableRasterобъекта. Этот метод выдастIllegalArgumentException, если значения цветового компонента для этого ColorModelне будут удобно представимы в ненормализованной форме. ArrayIndexOutOfBoundsExceptionВыбрасываетсяcomponents, если массив недостаточно велик, чтобы вместить все цветовые и альфа-компоненты (начиная с offset). Если objпеременная равна null, то будет выделен новый массив. Если objэто не nullтак, то это должен быть примитивный массив типа transferType; в противном случае ClassCastExceptionвыдается а. AnArrayIndexOutOfBoundsException выбрасывается, если objнедостаточно велико, чтобы содержать значение пикселя для этого ColorModel. Поскольку ColorModelэто абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны переопределять этот метод, поскольку реализация в этом абстрактном классе выдает UnsupportedOperationException.

Параметры:

components - массив ненормализованных цветовых и альфа-компонентов

offset - индекс, с componentsкоторого начинается извлечение цветовых и альфа-компонентов

obj - Objectпредставляет массив цветовых и альфа-компонентов

ВОЗВРАТ:

Objectпредставляет собой массив цветовых и альфа-компонентов.

Выдает:

ClassCastException - if objне является примитивным массивом типа transferType

ArrayIndexOutOfBoundsException - если objнедостаточно велико, чтобы содержать значение пикселя для этогоColorModel, или componentsмассив недостаточно велик, чтобы содержать все цветовые и альфа-компоненты, начиная с offset

IllegalArgumentException - если значения компонентов для этого ColorModelне могут быть удобно представлены в ненормализованной форме

UnsupportedOperationException - если этот метод не поддерживается этим ColorModel

Object **getDataElements**(int[] components, int offset, **Object** obj)

Возвращает массив элементов данных, представляющий пиксель в этом ColorModel, учитывая массив ненормализованных цветовых / альфа-компонентов.

Затем этот массив может быть передан setDataElementsметоду WritableRasterобъекта. ArrayIndexOutOfBoundsExceptionВыбрасываетсяnormComponents, если массив недостаточно велик, чтобы вместить все цветовые и альфа-компоненты (начиная с normOffset). Если objпеременная равна null, то будет выделен новый массив. Если objэто не nullтак, то это должен быть примитивный массив типа transferType; в противном случае ClassCastExceptionвыдается а. ArrayIndexOutOfBoundsExceptionВыбрасывается значение , если objоно недостаточно велико , чтобы содержать значение пикселя для этого ColorModel. Поскольку ColorModelэто абстрактный класс, любой экземпляр является экземпляром подкласса. Реализация этого метода по умолчанию в этом абстрактном классе сначала преобразует из нормализованной формы в ненормализованную форму, а затем вызывает getDataElement(int[], int, Object). Подклассы, которые могут иметь экземпляры, которые не поддерживают ненормализованную форму, должны переопределять этот метод.

Параметры:

normComponents - массив нормализованных цветовых и альфа-компонентов

normOffset - индекс, с normComponentsкоторого начинается извлечение цветовых и альфа-компонентов

obj - примитивный массив данных для хранения возвращенного пикселя

ВОЗВРАТ:

Objectкоторая представляет собой примитивное представление массива данных пикселя

Выдает:

ClassCastException - if objне является примитивным массивом типа transferType

ArrayIndexOutOfBoundsException - если objнедостаточно велико, чтобы содержать значение пикселя для этогоColorModel, или normComponentsмассив недостаточно велик, чтобы содержать все цветовые и альфа-компоненты, начиная с normOffset

Object **getDataElements**(int rgb, **Object** pixel)

Возвращает массив элементов данных, представляющий пиксель в этом ColorModel, учитывая целочисленное пиксельное представление в цветовой модели RGB по умолчанию.

Затем этот массив может быть передан WritableRaster.setDataElements(int, int, java.lang.Object) методу WritableRasterобъекта. Если переменная pixel nullравна , то будет выделен новый массив. Если pixelэто не nullтак, это должен быть примитивный массив типа transferType; в противном случае ClassCastExceptionвыдается а . ArrayIndexOutOfBoundsExceptionВыбрасывается значение , если pixelоно недостаточно велико , чтобы содержать значение пикселя для этого ColorModel. Возвращается массив пикселей. Если это transferTypeне поддерживается, UnsupportedOperationExceptionбудет выдан а . Поскольку ColorModelэто абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны переопределять этот метод , поскольку реализация в этом абстрактном классе выдает UnsupportedOperationException.

Параметры:

rgb - представление целых пикселей в цветовой модели RGB по умолчанию

pixel - указанный пиксель

ВОЗВРАТ:

представление массива указанного пикселя в этом ColorModel.

Выдает:

ClassCastException - if pixelне является примитивным массивом типа transferType

ArrayIndexOutOfBoundsException - если pixelнедостаточно велико, чтобы содержать значение пикселя для этого ColorModel

UnsupportedOperationException - если этот метод не поддерживается этим ColorModel

abstract int	**getGreen**(int pixel) Возвращает компонент зеленого цвета для указанного пикселя, масштабируемый от 0 до 255 в цветовом пространстве RGB по умолчанию, sRGB. При необходимости выполняется преобразование цвета. Значение пикселя указывается как значение int. Выбрасывается IllegalArgumentException, если значения пикселей для этого ColorModelне могут быть представлены в виде одного int. Возвращаемое значение не является предварительно умноженным значением. Например, если альфа-значение предварительно умножается, этот метод делит его, прежде чем возвращать значение. Если альфа-значение равно 0, то значение зеленого равно 0. **Параметры:** pixel - указанный пиксель **ВОЗВРАТ:** значение зеленой составляющей указанного пикселя.
int	**getGreen**(**Object** inData) Возвращает компонент зеленого цвета для указанного пикселя, масштабируемый от 0 до 255 в RGB по умолчанию ColorSpace, sRGB. При необходимости выполняется преобразование цвета. Значение пикселя определяется массивом элементов данных типа transferType, переданных в качестве ссылки на объект. Возвращаемое значение не будет предварительно умноженным значением. Например, если альфа-значение предварительно умножается, этот метод делит его, прежде чем возвращать значение. Если альфа-значение равно 0, то зеленое значение равно 0. Если inDataэто не примитивный массив типа transferType, ClassCastExceptionвыбрасывается а . АнArrayIndexOutOfBoundsException выбрасывается , если inDataон недостаточно велик , чтобы содержать значение пикселя для этого ColorModel. Если это transferTypeне поддерживается, UnsupportedOperationExceptionбудет выброшен символ а. Поскольку ColorModelэто абстрактный класс, любой экземпляр должен быть экземпляром подкласса. Подклассы наследуют реализацию этого метода, и если они не переопределяют его, этот метод генерирует исключение, если подкласс использует transferTypeзначение, отличное от DataBuffer.TYPE_BYTE, DataBuffer.TYPE_USHORT, или DataBuffer.TYPE_INT. **Параметры:** inData - массив значений пикселей **ВОЗВРАТ:** значение зеленой составляющей указанного пикселя. **Выбрасывает:** ClassCastException - если inData не является примитивным массивом типа transferType

	ArrayIndexOutOfBoundsException - если inData недостаточно велико, чтобы содержать значение пикселя для этого ColorModel UnsupportedOperationException - если это tranferType не подтверждается этим ColorModel
float[]	**getNormalizedComponents**(int[] components, int offset, float[] normComponents, int normOffset) Возвращает массив всех цветовых / альфа-компонентов в нормализованной форме, учитывая ненормализованный массив компонентов. Ненормализованные компоненты представляют собой целые значения без знака в диапазоне от 0 до $2^n - 1$, где n - количество битов для конкретного компонента. Нормализованные компоненты представляют собой значения с плавающей запятой между минимумом и максимумом для каждого компонента, заданными ColorSpaceобъектом для этого ColorModel. An IllegalArgumentExceptionбудет выдан, если значения цветового компонента для этого ColorModelне могут быть удобно представлены в ненормализованной форме. Если normComponentsмассив естьnull, то будет выделен новый массив. normComponentsБудет возвращен массив. Компоненты цвета / альфа хранятся в normComponentsмассиве, начиная с normOffset(даже если массив выделен этим методом). ArrayIndexOutOfBoundsExceptionВыбрасываетсяnormComponents, если массив не является nullи не является достаточно большим, чтобы содержать все цветовые и альфа-компоненты (начиная с normOffset). IllegalArgumentExceptionЕсли componentsмассив недостаточно велик, чтобы вместить все цветовые и альфа-компоненты, начиная с offset. Поскольку ColorModelэто абстрактный класс, любой экземпляр является экземпляром подкласса. Реализация этого метода по умолчанию в этом абстрактном классе предполагает, что значения компонентов для этого класса удобно представлять в ненормализованной форме. Следовательно, подклассы, которые могут иметь экземпляры, не поддерживающие ненормализованную форму, должны переопределять этот метод. **Параметры:** components - массив, содержащий ненормализованные компоненты offset - смещение в componentsмассив, с которого начинается извлечение ненормализованных компонентов normComponents - массив , который принимает нормализованные компоненты

normOffset - индекс, с normComponentsкоторого следует начинать хранение нормализованных компонентов

ВОЗВРАТ:

массив, содержащий нормализованные цветовые и альфа-компоненты.

Выдает:

IllegalArgumentException - Если значения компонентов для этого ColorModelне могут быть удобно представлены в ненормализованной форме.

UnsupportedOperationException - если конструктор этого ColorModelвызвал super(bits)конструктор, но не переопределил этот метод. См . Конструктор, ColorModel(int).

UnsupportedOperationException - если этот метод не может определить количество битов на компонент

float[]

getNormalizedComponents(**Object** pixel, float[] normComponents, int normOffset)

Возвращает массив всех цветовых / альфа-компонентов в нормализованной форме, учитывая пиксель в этом ColorModel.

Ненормализованные компоненты представляют собой целые значения без знака в диапазоне от 0 до 2 ⁿ - 1, где n - количество битов для конкретного компонента. Нормализованные компоненты представляют собой значения с плавающей запятой между минимумом и максимумом для каждого компонента, заданными ColorSpaceобъектом для этого ColorModel. An IllegalArgumentExceptionбудет выдан, если значения цветового компонента для этого ColorModelне могут быть удобно представлены в ненормализованной форме. Если normComponentsмассив естьnull, то будет выделен новый массив. normComponentsБудет возвращен массив. Компоненты цвета / альфа хранятся в normComponentsмассиве, начиная с normOffset(даже если массив выделен этим методом). ArrayIndexOutOfBoundsExceptionВыбрасываетсяnorm Components, если массив не является nullи не является достаточно большим, чтобы содержать все цветовые и альфа-компоненты (начиная с normOffset). IllegalArgumentExceptionЕсли componentsмасс ив недостаточно велик, чтобы вместить все цветовые и альфа-компоненты, начиная с offset.

Поскольку ColorModelэто абстрактный класс, любой экземпляр является экземпляром подкласса. Реализация этого метода по умолчанию в этом абстрактном классе предполагает, что значения компонентов для этого класса удобно представлять в ненормализованной форме. Следовательно, подклассы, которые

	могут иметь экземпляры, не поддерживающие ненормализованную форму, должны переопределять этот метод. **Параметры:** components - массив, содержащий ненормализованные компоненты offset - смещение в componentsмассив, с которого начинается извлечение ненормализованных компонентов normComponents - массив , который принимает нормализованные компоненты normOffset - индекс, с normComponents которого следует начинать хранение нормализованных компонентов **ВОЗВРАТ:** массив, содержащий нормализованные цветовые и альфа-компоненты. **Выдает:** IllegalArgumentException - Если значения компонентов для этого ColorModelне могут быть удобно представлены в ненормализованной форме. UnsupportedOperationException - если конструктор этого ColorModelвызвал super(bits)конструктор, но не переопределил этот метод. См . Конструктор, ColorModel(int). UnsupportedOperationException - если этот метод не может определить количество битов на компонент
int	**getNumColorComponents**() Возвращает количество цветовых компонентов в этом ColorModel. Количество возвращаемых компонентов ColorSpace.getNumComponents(). **ВОЗВРАТ:** количество цветовых компонентов в этом ColorModel.
int	**getNumComponents**() Возвращает количество компонентов, включая альфа, в этом ColorModel. Это равно количеству цветовых компонентов плюс один, если есть альфа-компонент. **ВОЗВРАТ:** количество компонентов в этом ColorModel
int	**getPixelSize**()

	Возвращает количество бит на пиксель, описанное этим `ColorModel`. **ВОЗВРАТ:** `количество бит на пиксель.`

`abstract int`	**getRed**`(int pixel)`
	
	Возвращает компонент красного цвета для указанного пикселя, масштабируемый от 0 до 255 в цветовом пространстве RGB по умолчанию, sRGB.
	
	При необходимости выполняется преобразование цвета. Значение пикселя указывается как значение int. Выбрасывается `IllegalArgumentException`, если значения пикселей для этого `ColorModel` не могут быть представлены в виде одного int. Возвращаемое значение не является предварительно умноженным значением. Например, если альфа-значение предварительно умножается, этот метод делит его, прежде чем возвращать значение. Если альфа-значение равно 0, то красное значение равно 0.
	
	Параметры:
	`pixel - указанный пиксель`
	
	ВОЗВРАТ:
	`значение красной составляющей указанного пикселя.`

`int`	**getRed**`(`**Object** `inData)` Возвращает компонент красного цвета для указанного пикселя, масштабируемый от 0 до 255 в RGB по умолчанию `ColorSpace`, sRGB. При необходимости выполняется преобразование цвета. Значение пикселя задается массивом элементов данных типа transferType, передаваемых в качестве ссылки на объект. Возвращаемое значение не является предварительно умноженным значением. Например, если альфа предварительно умножается, этот метод делит ее, прежде чем возвращать значение. Если альфа-значение равно 0, то красное значение равно 0. Если `inData` это не примитивный массив типа transferType, то выбрасывается `ClassCastException`. `ArrayIndexOutOfBoundsException` выбра сывается , если `inData`он недостаточно велик , чтобы содержать значение пикселя для этого `ColorModel`. Если этот `transferType` не поддерживается, выбрасывается `UnsupportedOperationException`. Поскольку `ColorModel` это абстрактный класс, любой экземпляр должен быть экземпляром подкласса. Подклассы наследуют реализацию этого метода, и, если они не переопределяют его, этот метод генерирует исключение, если подкласс использует значение `transferType`, отличное от `DataBuffer.TYPE_BYTE`, `DataBuffer.TYPE_USHORT`, или `DataBuffer.TYPE_INT`. **Параметры:** `inData - массив значений пикселей` **ВОЗВРАТ:**

значение красной составляющей указанного пикселя.

Выбрасывает:

ClassCastException - if inDatane является примитивным массивом типа transferType

ArrayIndexOutOfBoundsException - если inDatanедостаточно велико, чтобы содержать значение пикселя для этого ColorModel

UnsupportedOperationException - если это tranferTypene подтверждается этим ColorModel

int **getRGB**(int pixel)

Возвращает цвет / альфа-компоненты пикселя в формате цветовой модели RGB по умолчанию.

При необходимости выполняется преобразование цвета. Значение пикселя указывается как int. Выбрасывается IllegalArgumentException если значения пикселей для этого ColorModelне могут быть представлены в виде одного int. Возвращаемое значение находится в формате без предварительного умножения. Например, если альфа-код предварительно умножается, этот метод разделяет его на цветовые компоненты. Если значение альфа равно 0, то значения цвета равны 0.

Параметры:

pixel - указанный пиксель

ВОЗВРАТ:

значение RGB для цветовых / альфа-компонентов указанного пикселя.

int **getRGB**(**Object** inData)

Возвращает цветовые /альфа-компоненты для указанного пикселя в формате цветовой модели RGB по умолчанию.

При необходимости выполняется преобразование цвета. Значение пикселя определяется массивом элементов данных типа transferType, переданных в качестве ссылки на объект. Если InData не является примитивным массивом типа transferType, ClassCastExceptionвыбрасывается a . ВыбрасываетсяArrayIndexOutOfBoundsException, если inDataзначение недостаточно велико, чтобы содержать значение пикселя для этого ColorModel. Возвращаемое значение будет в формате без предварительного умножения, т. Е. Если альфа-значение предварительно умножается, этот метод разделит его на цветовые компоненты (если значение альфа равно 0, значения цвета будут равны 0).

Параметры:

inData - указанный пиксель

ВОЗВРАТ:

цвет и альфа-компоненты указанного пикселя.

static **ColorModel**	**getRGBdefault**() Возвращает DirectColorModel это описывает формат по умолчанию для целых значений RGB, используемых во многих методах в интерфейсах изображений AWT для удобства программиста. Цветовое пространство используется по умолчаниюColorSpace, sRGB. Формат значений RGB представляет собой целое число с 8 битами каждого из компонентов альфа, красного, зеленого и синего цветов, упорядоченных соответственно от старшего значащего байта до младшего значащего байта, как в: 0xAARRGGBB . Компоненты цвета не умножаются предварительно альфа-компонентом. Этот формат не обязательно представляет собственный или наиболее эффективный ColorModel для конкретного устройства или для всех изображений. Он просто используется как общий формат цветовой модели. **ВОЗВРАТ:** DirectColorModelобъект, описывающий значения RGB по умолчанию.
int	**getTransferType**() Возвращает тип передачи этого ColorModel. Тип передачи - это тип примитивного массива, используемого для представления значений пикселей в виде массивов. **ВОЗВРАТ:** тип передачи.
int	**getTransparency**() Возвращает прозрачность. **ВОЗВРАТ:** прозрачность этого ColorModel: Transparency.OPAQUE, Transparency.BITMASK или Transparency.TRANSLUCENT
int[]	**getUnnormalizedComponents**(float[] normComponents, int normOffset, int[] components, int offset) Возвращает массив всех цветовых/альфа-компонентов в ненормализованной форме, учитывая нормализованный массив компонентов. Ненормализованные компоненты представляют собой целочисленные значения без знака в диапазоне от 0 до $2^n - 1$, где n - количество битов для конкретного компонента. Нормализованные компоненты - это значения с плавающей запятой между минимумом и максимумом для каждого компонента , заданными ColorSpaceобъектом для этого ColorModel. An IllegalArgumentExceptionбудет выдан, если значения цветового компонента для этого ColorModelне могут быть удобно представлены в ненормализованной форме. Если componentsмассив естьnull, то будет выделен новый массив. componentsМассив будет возвращен. Цветовые / альфа-

компоненты хранятся в `components`массиве, начиная с `offset`(даже если массив выделен этим
методом). `ArrayIndexOutOfBoundsException`Выбрасывается`components`, если массив не является `null`и не является достаточно большим, чтобы вместить все цветовые и альфа-компоненты (начиная
с`offset`). `IllegalArgumentException`Если `normComponents`массив недостаточно велик, чтобы вместить все цветовые и альфа-компоненты, начиная с `normOffset`.

Параметры:

`normComponents` - массив , содержащий нормализованные компоненты

`normOffset` - смещение в `normComponents`массив, с которого начинается извлечение нормализованных компонентов

`components` - массив, который получает компоненты из `normComponents`

`offset` - индекс, с `components`которого следует начинать хранение нормализованных компонентов, начиная с `normComponents`

ВОЗВРАТ:

массив, содержащий ненормализованные цветовые и альфа-компоненты.

Выдает:

`IllegalArgumentException` - Если значения компонентов для этого `ColorModel`не могут быть удобно представлены в ненормализованной форме.

`IllegalArgumentException` - если длина `normComponents`минус `normOffset`меньше, чем `numComponents`

`UnsupportedOperationException` - если конструктор этого `ColorModel`вызвал `super(bits)`конструктор, но не переопределил этот метод. См . Конструктор, `ColorModel(int)`.

`boolean`	**`hasAlpha`**`()` Возвращает, поддерживается ли альфа-код в этом `ColorModel`. **ВОЗВРАТ:** `true` если в этом `ColorModel` поддерживается альфа; `false` в противном случае.
`int`	**`hashCode`**`()` Возвращает хэш-код для этой цветовой модели.
`boolean`	**`isAlphaPremultiplied`**`()`

	Возвращает, был ли альфа-символ предварительно умножен в значениях пикселей, которые будут преобразованы этим `ColorModel`. Если логическое значение равно `true`, оно `ColorModel` должно использоваться для интерпретации значений пикселей, в которых информация о цвете и альфа представлены в виде отдельных пространственных полос, и предполагается, что образцы цвета были умножены на альфа-образец. **ВОЗВРАТ:** `true если альфа-значения предварительно умножаются на значения пикселей, которые будут преобразованы этим ColorModel; false в противном случае.`

`boolean`	**`isCompatibleRaster`**`(`**`Raster`** `raster)`
	
	ВОЗВРАТ `true` если `raster` совместим с этим `ColorModel` и `false` если это не так.
	
	Поскольку `ColorModel`это абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны переопределять этот метод, поскольку реализация в этом абстрактном классе выдает `UnsupportedOperationException`.
	
	Параметры:
	`raster - Rasterобъект для проверки на совместимость`
	
	ВОЗВРАТ:
	`true если rasterэто совместимо с этим ColorModel.`
	
	Выдает:
	`UnsupportedOperationException - если этот метод не был реализован для этого ColorModel`

`boolean`	**`isCompatibleSampleModel`**`(`**`SampleModel`** `sm)`
	
	Проверяет, является ли `SampleModel` совместим с этим `ColorModel`.
	
	Поскольку `ColorModel`это абстрактный класс, любой экземпляр является экземпляром подкласса. Подклассы должны переопределять этот метод, поскольку реализация в этом абстрактном классе выдает `UnsupportedOperationException`.
	
	Параметры:
	`sm - указанный SampleModel`
	
	ВОЗВРАТ:
	`true если указанное SampleModelсовместимо с этимColorModel; falseв противном случае.`
	
	Выдает:
	`UnsupportedOperationException - если этот метод не поддерживается этим ColorModel`

String	toString() Возвращает String представление содержимого этого ColorModelобъект.

Класс ComponentColorModel

ColorModelКласс, который работает со значениями пикселей, которые представляют информацию о цвете и альфа-канале в виде отдельных выборок и хранят каждую выборку в отдельном элементе данных. Этот класс может использоваться с произвольным ColorSpace. Количество цветовых выборок в значениях пикселей должно совпадать с количеством цветовых компонентов в ColorSpace. Может быть один альфа-образец.

Для тех методов, которые используют представление типа примитивного массива в пикселях transferType, длина массива равна количеству цветовых и альфа-выборок. Образцы цветов сохраняются сначала в массиве, за которым следует альфа-образец, если он присутствует. Порядок цветовых образцов определяется ColorSpace. Обычно этот порядок отражает имя типа цветового пространства. Например, для TYPE_RGB, индекс 0 соответствует красному, индекс 1 - зеленому, а индекс 2 - синему.

Преобразование значений выборки пикселей в цветовые / альфа-компоненты для целей отображения или обработки основано на взаимно однозначном соответствии выборок компонентам. В зависимости от типа передачи, используемого для создания экземпляра ComponentColorModel, значения выборки пикселей, представленные этим экземпляром, могут быть подписанными или беззнаковыми и могут иметь целочисленный тип, плавающий или двойной (подробности см. Ниже). Преобразование значений выборки в нормализованные цветовые / альфа-компоненты должно соответствовать определенным правилам. Для выборок с плавающей запятой и double преобразование является идентификатором, т.Е.

нормализованные значения компонентов равны соответствующим значениям выборки. Для целых выборок преобразование должно быть только простым масштабом и смещением, где константы масштаба и смещения могут отличаться для каждого компонента. Результатом применения констант масштаба и смещения является набор значений цветовых / альфа-компонентов, которые гарантированно попадают в определенный диапазон. Как правило, диапазон для цветового компонента будет диапазоном, определенным методами `getMinValue`и `getMaxValueColorSpace`класса. Диапазон для альфа-компонента должен быть от 0.0 до 1.0.

Экземпляры `ComponentColorModel`, созданные с типами передачи `DataBuffer.TYPE_BYTE`, `DataBuffer.TYPE_USHORT`, и `DataBuffer.TYPE_INT`имеют значения выборки пикселей, которые обрабатываются как целые значения без знака. Количество битов в цветовой или альфа-выборке значения пикселя может не совпадать с количеством битов для соответствующего цвета или альфа-выборки, переданных `ComponentColorModel(ColorSpace, int[], boolean, boolean, int, int)` конструктору. В этом случае этот класс предполагает, что n младших значащих битов значения выборки содержат значение компонента, где n - количество значащих битов для компонента, переданного конструктору. Он также предполагает, что любые биты более высокого порядка в значении выборки равны нулю. Таким образом, значения выборки варьируются от 0 до $2^n - 1$. Этот класс сопоставляет эти значения выборки с нормализованными значениями цветового компонента таким образом, что 0 соответствует значению, полученному из `ColorSpace's getMinValue`метода для каждого компонента, а $2^n - 1$ соответствует значению, полученному из `getMaxValue`. Для создания сопоставления `ComponentColorModel` с образцом другого цвета требуется подкласс этого класса и

переопределение `getNormalizedComponents(Object, float[], int)` метода. Отображение для альфа-выборки всегда отображает 0 в 0.0 и $2^n - 1$ в 1.0.

Для экземпляров с неподписанными значениями выборки ненормализованное представление цвета / альфа-компонента поддерживается только при соблюдении двух условий. Во-первых, значение выборочного значения 0 должно соответствовать нормализованному значению компонента 0.0, а значение выборки $2^n - 1$ до 1.0. Во-вторых, минимальный / максимальный диапазон всех цветовых компонентов `ColorSpace`должен быть от 0.0 до 1.0. В этом случае представление компонента представляет собой n младших значащих битов соответствующей выборки. Таким образом, каждый компонент представляет собой целое значение без знака в диапазоне от 0 до $2^n - 1$, где n - количество значащих битов для конкретного компонента. Если эти условия не выполняются, любой метод, принимающий ненормализованный аргумент компонента, выдаст `IllegalArgumentException`.

Экземпляры `ComponentColorModel`, созданные с типами передачи `DataBuffer.TYPE_SHORT`, `DataBuffer.TYPE_FLOAT`, и `DataBuffer.TYPE_DOUBLE`имеют значения выборки пикселей, которые обрабатываются как короткие, плавающие или двойные значения со знаком. Такие экземпляры не поддерживают ненормализованное представление цвета / альфа-компонента, поэтому любые методы, принимающие такое представление в качестве аргумента, будут выдавать `IllegalArgumentException`при вызове в одном из этих экземпляров. Нормализованные значения компонентов экземпляров этого класса имеют диапазон, который зависит от типа передачи следующим образом: для выборок с плавающей запятой - полный диапазон типа данных с плавающей запятой; для двойных выборок - полный диапазон типа данных с плавающей запятой

(полученный в результате преобразования double в float); для коротких выборок - отприблизительно от -maxVal до +maxVal, где maxVal - максимальное значение для каждого компонента `ColorSpace`для (-32767 сопоставляется с -maxVal, 0 сопоставляется с 0.0, а 32767 сопоставляется с +maxVal). Подкласс может переопределить масштабирование для коротких выборочных значений до нормализованных значений компонента путем переопределения `getNormalizedComponents(Object, float[], int)` метода. Для выборок с плавающей запятой и double нормализованные значения компонентов принимаются равными соответствующим значениям выборки, и подклассы не должны пытаться добавлять какое-либо неидентичное масштабирование для этих типов передачи.

Экземпляры `ComponentColorModel`создаются с типами передачи `DataBuffer.TYPE_SHORT`, `DataBuffer.TYPE_FLOAT`, и `DataBuffer.TYPE_DOUBLE`используют все биты всех выборочных значений. Таким образом, все цветовые / альфа-компоненты имеют 16 бит при использовании`DataBuffer.TYPE_SHORT`, 32 бита при использовании `DataBuffer.TYPE_FLOAT`и 64 бита при использовании `DataBuffer.TYPE_DOUBLE`. Когда `ComponentColorModel(ColorSpace, int[], boolean, boolean, int, int)` форма конструктора используется с одним из этих типов передачи, аргумент массива bits игнорируется.

Возможно иметь значения выборки цвета / альфа, которые нельзя разумно интерпретировать как значения компонентов для рендеринга. Это может произойти, когда `ComponentColorModel`подкласс is переопределяет сопоставление значений выборки без знака с нормализованными значениями цветового компонента или когда используются значения выборки со знаком за пределами определенного диапазона. (В качестве

примера, указание альфа-компонента в качестве короткого значения со знаком за пределами диапазона от 0 до 32767, нормализованного диапазона от 0.0 до 1.0, может привести к неожиданным результатам.) Приложения несут ответственность за надлежащее масштабирование пиксельных данных перед рендерингом таким образом, чтобы цветовые компоненты попадали в нормализованный диапазон `ColorSpace`(полученный с использованием методов `getMinValue`и `getMaxValueColorSpace`класса), а альфа-компонент находился в диапазоне от 0.0до 1.0. Если значения цвета или альфа-компонента выходят за пределы этих диапазонов, результаты рендеринга не определены.

Методы, использующие представление одного пикселя int, выдают `IllegalArgumentException`значение , если только число компонентов для параметра `ComponentColorModel` равно единице, а значение компонента не имеет знака - другими словами, компонент с одним цветом, использующий тип передачи `DataBuffer.TYPE_BYTE`, `DataBuffer.TYPE_USHORT`, или `DataBuffer.TYPE_INT` и без альфа.

А `ComponentColorModel` может использоваться в сочетании с а `ComponentSampleModel`, `BandedSampleModel` или `PixelInterleavedSampleModel` для построения а `BufferedImage`.

Конструкторы ComponentColorModel

Конструктор и описание
`ComponentColorModel`(**`ColorSpace`** `colorSpace, boolean hasAlpha, boolean isAlphaPremultiplied, int transparency, int transferType)` Создает а `ComponentColorModel`из указанных параметров. Цвет компонентов будет в указанном `ColorSpace`. Поддерживаемые типы передачи `DataBuffer.TYPE_BYTE`, `DataBuffer.TYPE_USHORT`, `DataBuffer.TYPE_INT`, `DataBuffer.TYPE_SHORT`, `DataBuffer.TYPE_FLOAT`, и `DataBuffer.TYPE_DOUBLE`. Если значение не равно null, `bits`массив указывает количество значащих битов на цвет и альфа-компонент, и его длина должна быть не менее количества компонентов в`ColorSpace`, если

в значениях пикселей нет альфа-информации, или на единицу больше этого числа, если есть альфа-информация. Когда аргумент `transferType`is `DataBuffer.TYPE_SHORT`, `DataBuffer.TYPE_FLOAT`, или `DataBuffer.TYPE_DOUBLE`bitsarray игнорируется. `hasAlpha`указывает, присутствует ли альфа-информация. Если`hasAlpha` имеет значение true, тогда логическое `isAlphaPremultiplied`значение указывает, как интерпретировать цветовые и альфа-выборки в значениях пикселей. Если логическое значение равно true, предполагается, что образцы цвета были умножены на альфа-образец. `transparency`Указывает, какие альфа-значения могут быть представлены этой цветовой моделью. Допустимыми `transparency`значениями являются `OPAQUE`, `BITMASK`или `TRANSLUCENT`. Это `transferType`тип примитивного массива, используемого для представления значений пикселей.

Параметры:

`colorSpace` - `ColorSpace`Связанная с этой цветовой моделью.

`bits` - Количество значащих битов на компонент. Может иметь значение null, и в этом случае все биты всех выборок компонентов будут значимыми. Игнорируется, если transferType является одним из DataBuffer.TYPE_SHORT, DataBuffer.TYPE_FLOAT, или DataBuffer.TYPE_DOUBLE, и в этом случае все биты всех выборок компонентов будут значимыми.

`hasAlpha` - Если true, эта цветовая модель поддерживает альфа-версию.

`isAlphaPremultiplied` - Если значение true, альфа умножается предварительно.

`transparency` - Указывает, какие альфа-значения могут быть представлены этой цветовой моделью.

`transferType` - Определяет тип примитивного массива, используемого для представления значений пикселей.

Выбрасывает:

`IllegalArgumentException` - Если аргумент bits массива не равен null, его длина меньше, чем количество цветовых и альфа-компонентов, а transferType является одним из DataBuffer.TYPE_BYTE, DataBuffer.TYPE_USHORT, или DataBuffer.TYPE_INT.

`IllegalArgumentException` - Если transferType не является одним из DataBuffer.TYPE_BYTE, DataBuffer.TYPE_USHORT, DataBuffer.TYPE_INT, DataBuffer.TYPE_SHORT, DataBuffer.TYPE_FLOAT, или DataBuffer.TYPE_DOUBLE.

`ComponentColorModel`(**`ColorSpace`** `colorSpace, int[] bits, boolean hasAlpha, boolean isAlphaPremultiplied, int transparency, int transferType`)

Создает a `ComponentColorModel`из указанных параметров.

Поддерживаемые типы передачи `DataBuffer.TYPE_BYTE`, `DataBuffer.TYPE_USHORT`, `DataBuffer.TYPE_INT`, `DataBuffer.TYPE_SHORT`, `DataBuffer.TYPE_FLOAT`, и `DataBuffer.TYPE_DOUBLE`. Количество значащих битов для каждого цветового и альфа-компонента будет 8, 16, 32, 16, 32, или 64, соответственно. Количество цветовых

компонентов будет равно количеству компонентов в `ColorSpace`. Если есть, будет альфа-компонент `hasAlphatrue`. Если `hasAlpha`значение равно true, то логическое `isAlphaPremultiplied`значение указывает, как интерпретировать цветовые и альфа-выборки в значениях пикселей. Если логическое значение равно true, предполагается, что образцы цвета были умножены на альфа-образец. `transparency`Указывает, какие альфа-значения могут быть представлены этой цветовой моделью. Допустимыми `transparency`значениями являются `OPAQUE`, `BITMASK`или `TRANSLUCENT`. Это `transferType`тип примитивного массива, используемого для представления значений пикселей.

Параметры:

`colorSpace - ColorSpaceСвязанная с этой цветовой моделью.`

`hasAlpha - Если true, эта цветовая модель поддерживает альфа-версию.`

`isAlphaPremultiplied - Если значение true, альфа умножается предварительно.`

`transparency - Указывает, какие альфа-значения могут быть представлены этой цветовой моделью.`

`transferType - Определяет тип примитивного массива, используемого для представления значений пикселей.`

Выдает:

`IllegalArgumentException - Если transferType не является одним из DataBuffer.TYPE_BYTE, DataBuffer.TYPE_USHORT, DataBuffer.TYPE_INT, DataBuffer.TYPE_SHORT, DataBuffer.TYPE_FLOAT, или DataBuffer.TYPE_DOUBLE.`

Класс IndexColorModel

`IndexColorModel`Класс - это `ColorModel`класс, который работает со значениями пикселей, состоящими из одного образца, который является индексом в фиксированной цветовой карте в цветовом пространстве sRGB по умолчанию. Цветовая карта определяет красный, зеленый, синий и необязательные альфа-компоненты, соответствующие каждому индексу. Все компоненты представлены в цветовой карте в виде 8-разрядных целых значений без знака. Некоторые конструкторы позволяют вызывающей стороне указывать "дыры" в цветовой карте, указывая, какие элементы цветовой карты допустимы, а какие представляют непригодные цвета с помощью битов, установленных в `BigInteger`объекте. Эта цветовая модель похожа на псевдоцветное визуальное изображение X11.

Некоторые конструкторы предоставляют средства для указания альфа-компонента для каждого пикселя в цветовой карте, в то время как другие либо не предоставляют таких средств, либо, в некоторых случаях, флаг, указывающий, содержат ли данные цветовой карты альфа-значения. Если в конструктор не задана альфа-версия, для каждой записи предполагается непрозрачный альфа-компонент (alpha = 1.0). Может быть предоставлено необязательное значение прозрачного пикселя, которое указывает, что пиксель должен быть полностью прозрачным, независимо от любого альфа-компонента, предоставленного или предполагаемого для этого значения пикселя. Обратите внимание, что цветовые компоненты в цветовой `IndexColorModel`карте объектов никогда предварительно не умножаются на альфа-компоненты.

Прозрачность объекта `IndexColorModel` определяется путем изучения альфа-компонентов цветов в цветовой карте и выбора наиболее конкретного значения после рассмотрения необязательных альфа-значений и любого указанного прозрачного индекса. Значение прозрачности `Transparency.OPAQUE`задается только в том случае, если все допустимые цвета в цветовой карте непрозрачны и нет допустимого прозрачного пикселя. Если все допустимые цвета в цветовой карте либо полностью непрозрачны (альфа = 1,0), либо полностью прозрачны (альфа = 0,0), что обычно происходит, когда указан допустимый прозрачный пиксель, значение равно `Transparency.BITMASK`. В противном случае значение равно `Transparency.TRANSLUCENT`, указывая, что некоторый допустимый цвет имеет альфа-компонент, который не является ни полностью прозрачным, ни полностью непрозрачным (0.0 < alpha < 1.0).

Если `IndexColorModel` объект имеет значение прозрачности равное `Transparency.OPAQUE`, то

методы `hasAlpha`и `getNumComponents` (оба унаследованы от `ColorModel`) возвращают false и 3 соответственно. Для любого другого значения прозрачности `hasAlpha`возвращает true и `getNumComponents`возвращает 4 .

Значения, используемые для индексации в цветовой карте, берутся из *n* младших значащих битов пиксельных представлений, где *n* основано на размере пикселя, указанном в конструкторе. Для размеров пикселей, меньших 8 бит, *n* округляется до степени двух (3 становится 4, а 5,6,7 становится 8). Для размеров пикселей от 8 до 16 бит *n* равно размеру пикселя. Размеры пикселей, превышающие 16 бит, не поддерживаются этим классом. Биты более высокого порядка, чем *n*, игнорируются в пиксельных представлениях. Значения индекса больше или равны размеру карты, но меньше 2^n, не определены и возвращают 0 для всех цветовых и альфа-компонентов.

Для тех методов, которые используют представление типа примитивного массива в пикселях `transferType`, длина массива всегда равна единице. Поддерживаемые типы передачи`DataBuffer.TYPE_BYTE`: и `DataBuffer.TYPE_USHORT`. Для всех объектов этого класса допустимо представление одного пикселя int, поскольку всегда можно представить значения пикселей, используемые с этим классом, в одном int. Поэтому методы, использующие это представление, не выдают начение `IllegalArgumentException`з из-за недопустимого значения пикселя.

Многие методы в этом классе являются окончательными. Причина этого заключается в том, что базовый собственный графический код делает предположения о компоновке и работе этого класса, и эти предположения отражены в реализациях методов здесь, которые отмечены как окончательные. Вы можете

создать подкласс этого класса по другим причинам, но вы не можете переопределить или изменить поведение этих методов.

Конструкторы класса `IndexColorModel`

Конструктор и описание

`IndexColorModel``(int bits, int size, byte[] r, byte[] g, byte[] b)`
 	
Создает `IndexColorModel` из указанных массивов красных, зеленых и синих компонентов.
 	
Все пиксели, описываемые этой цветовой моделью, имеют альфа-компоненты 255 ненормализованных (1,0 нормализованных), что означает, что они *полностью непрозрачны*. Все массивы, определяющие цветовые компоненты, должны содержать как минимум указанное количество записей. Это `ColorSpace`пространство sRGB по умолчанию. Поскольку ни в одном из аргументов этого конструктора нет альфа-информации, значение прозрачности всегда равно `Transparency.OPAQUE`. Тип передачи является наименьшим из `DataBuffer.TYPE_BYTE` или `DataBuffer.TYPE_USHORT`, который может содержать один пиксель.
 	
Параметры:
 	
`bits - количество битов, которые занимает каждый пиксель`
 	
`size - размер массивов цветовых компонентов`
 	
`r - массив компонентов красного цвета`
 	
`g - массив компонентов зеленого цвета`
 	
`b - массив компонентов синего цвета`
 	
Выбрасывает:
`IllegalArgumentException - если bitsменьше 1 или больше 16`
 	
`IllegalArgumentException - если sizeменьше 1`

`IndexColorModel``(int bits, int size, byte[] r, byte[] g, byte[] b, byte[] a)`
 	
Создает `IndexColorModel`из заданных массивов красного, зеленого, синего и альфа-компонентов.
 	
Все массивы, определяющие компоненты, должны содержать как минимум указанное количество записей. Это `ColorSpace` пространство sRGB по умолчанию. Значение прозрачности может быть любым из `Transparency.OPAQUE`, `Transparency.BITMASK`, или `Transparency.TRANSLUCENT`в зависимости от аргументов, как указано в описании класса выше. Тип передачи является наименьшим из `DataBuffer.TYPE_BYTE` или `DataBuffer.TYPE_USHORT`, который может содержать один пиксель.
 	
Параметры:
 	
`bits - количество битов, которые занимает каждый пиксель`
 	
`size - размер массивов цветовых компонентов`
 	
`r - массив компонентов красного цвета`
 	
`g - массив компонентов зеленого цвета`
 	
`b - массив компонентов синего цвета`

a - массив компонентов с альфа-значениями

Выдает:

IllegalArgumentException - если bitsменьше 1 или больше 16

IllegalArgumentException - если sizeменьше 1

IndexColorModel(int bits, int size, byte[] r, byte[] g, byte[] b, int trans)

Создает IndexColorModelиз заданных массивов красных, зеленых и синих компонентов.

Все пиксели, описанные этой цветовой моделью, имеют альфа-компоненты 255 ненормализованных (1,0 нормализованных), что означает, что они полностью непрозрачны, за исключением того, что указанный пиксель должен быть сделан прозрачным. Все массивы, определяющие цветовые компоненты, должны содержать как минимум указанное количество записей. Это ColorSpaceпространство sRGB по умолчанию. Значение прозрачности может быть Transparency.OPAQUE или Transparency.BITMASK в зависимости от аргументов, как указано в описании класса выше. Тип передачи является наименьшим из DataBuffer.TYPE_BYTE или DataBuffer.TYPE_USHORT, который может содержать один пиксель.

Параметры:

bits - количество битов, которые занимает каждый пиксель

size - размер массивов цветовых компонентов

r - массив компонентов красного цвета

g - массив компонентов зеленого цвета

b - массив компонентов синего цвета

trans - индекс прозрачного пикселя

Выбрасывает:

IllegalArgumentException - если bitsменьше 1 или больше 16

IllegalArgumentException - если sizeменьше 1

IndexColorModel(int bits, int size, byte[] cmap, int start, boolean hasalpha)

Создает IndexColorModelиз одного массива чередующихся красных, зеленых, синих и необязательных альфа-компонентов.

В массиве должно быть достаточно значений, чтобы заполнить все необходимые массивы компонентов указанного размера. Это ColorSpace пространство sRGB по умолчанию. Значение прозрачности может быть любым из Transparency.OPAQUE, Transparency.BITMASK , или Transparency.TRANSLUCENT в зависимости от аргументов, как указано в описании класса выше. Тип передачи является наименьшим из DataBuffer.TYPE_BYTEилиDataBuffer.TYPE_USHORT , который может содержать один пиксель.

Параметры:

bits - количество битов, которые занимает каждый пиксель

size - размер массивов цветовых компонентов

cmap - массив цветовых компонентов

start - начальное смещение первого цветового компонента

hasalpha - указывает, содержатся ли альфа-значения в cmarмассиве

Выдает:

IllegalArgumentException - если bitsменьше 1 или больше 16

IllegalArgumentException - если sizeменьше 1

IndexColorModel(int bits, int size, byte[] cmap, int start, boolean hasalpha, int trans)

Создает IndexColorModel из одного массива чередующихся красных, зеленых, синих и необязательных альфа-компонентов.

Указанный прозрачный индекс представляет пиксель, который сделан полностью прозрачным, независимо от любого значения альфа, указанного для него. В массиве должно быть достаточно значений, чтобы заполнить все необходимые массивы компонентов указанного размера. Это ColorSpaceпространство sRGB по умолчанию. Значение прозрачности может быть любым из Transparency.OPAQUE, Transparency.BITMASK , или Transparency.TRANSLUCENT в зависимости от аргументов, как указано в описании класса выше. Тип передачи является наименьшим из DataBuffer.TYPE_BYTE или DataBuffer.TYPE_USHORT , который может содержать один пиксель.

Параметры:

bits - количество битов, которые занимает каждый пиксель

size - размер массивов цветовых компонентов

cmap - массив цветовых компонентов

start - начальное смещение первого цветового компонента

hasalpha - указывает, содержатся ли альфа-значения в cmarмассиве

trans - индекс полностью прозрачного пикселя

Выбрасывает:

IllegalArgumentException - если bitsменьше 1 или больше 16

IllegalArgumentException - если sizeменьше 1

IndexColorModel(int bits, int size, int[] cmap, int start, boolean hasalpha, int trans, int transferType)

Создает IndexColorModelиз массива целых чисел, где каждое целое число состоит из красного, зеленого, синего и необязательных альфа-компонентов в формате цветовой модели RGB по умолчанию.

Указанный прозрачный индекс представляет пиксель, который сделан полностью прозрачным, независимо от любого значения альфа, указанного для него. В массиве должно быть достаточно значений, чтобы заполнить все необходимые массивы компонентов указанного размера. Это ColorSpaceпространство sRGB по умолчанию. Значение прозрачности может быть любым из Transparency.OPAQUE, Transparency.BITMASK, или Transparency.TRANSLUCENTв зависимости от аргументов, как указано в описании класса выше.

Параметры:

bits - количество битов, которые занимает каждый пиксель

size - размер массивов цветовых компонентов

cmap - массив цветовых компонентов

start - начальное смещение первого цветового компонента

hasalpha - указывает, содержатся ли альфа-значения в cmapмассиве

trans - индекс полностью прозрачного пикселя

transferType - тип данных массива, используемого для представления значений пикселей. Тип данных должен быть либо DataBuffer.TYPE_BYTEили DataBuffer.TYPE_USHORT.

Выбрасывает:

IllegalArgumentException - если bitsменьше 1 или больше 16

IllegalArgumentException - если sizeменьше 1

IllegalArgumentException - если transferType не является одним из DataBuffer.TYPE_BYTE или DataBuffer.TYPE_USHORT

IndexColorModel(int bits, int size, int[] cmap, int start, int transferType, **BigInteger** validBits)

Создает IndexColorModelиз intмассива, каждый intиз которых состоит из красного, зеленого, синего и альфа-компонентов в формате цветовой модели RGB по умолчанию.

В массиве должно быть достаточно значений, чтобы заполнить все необходимые массивы компонентов указанного размера. Это ColorSpaceпространство sRGB по умолчанию. Значение прозрачности может быть любым из Transparency.OPAQUE, Transparency.BITMASK , или Transparency.TRANSLUCENTв зависимости от аргументов, как указано в описании класса выше. Тип передачи должен быть одним из DataBuffer.TYPE_BYTE DataBuffer.TYPE_USHORT. BigIntegerОбъект указывает допустимые / недопустимые пиксели в cmapмассиве. Пиксель допустим, если BigIntegerзадано значение в этом индексе, и недопустим, если BigInteger бит в этом индексе не установлен.

Параметры:

bits - количество битов, которые занимает каждый пиксель

size - размер массива цветовых компонентов

cmap - массив цветовых компонентов

start - начальное смещение первого цветового компонента

transferType - указанный тип данных

validBits - объект BigInteger. Если в BigInteger установлен бит, пиксель в этом индексе является допустимым. Если бит не установлен, пиксель с этим индексом считается недопустимым. Если значение равно null, все пиксели являются допустимыми. Учитываются только биты от 0 до размера карты.

Выдает:

IllegalArgumentException - если bitsменьше 1 или больше 16

IllegalArgumentException - если sizeменьше 1

```
IllegalArgumentException - если transferTypene является одним
из DataBuffer.TYPE_BYTEили DataBuffer.TYPE_USHORT
```

Класс PackedColorModel

Класс `PackedColorModel` представляет собой *абстрактный* класс `ColorModel`, который работает со значениями пикселей, которые представляют цветовую и альфа-информацию в виде отдельных выборок и которые упаковывают все выборки для одного пикселя в один int, short или байтовое количество. Этот класс может использоваться с произвольным `ColorSpace`. Количество цветовых выборок в значениях пикселей должно совпадать с количеством цветовых компонентов в `ColorSpace`. Может быть один альфа-образец. Длина массива всегда равна 1 для тех методов, которые используют примитивное представление типа в пикселях массива `transferType`. Поддерживаемые типы передачи - DataBuffer.TYPE_БАЙТ, DataBuffer.TYPE_USHORT и DataBuffer. TYPE_INT. Цветовые и альфа-образцы хранятся в одном элементе массива в битах, обозначенных битовыми масками. Каждая битовая маска должна быть смежной, и маски не должны перекрываться. Те же маски применяются к однопиксельному представлению int, используемому другими методами. Соответствие масок и образцов цвета / альфа выглядит следующим образом:

- Маски идентифицируются с помощью индексов, идущих от 0 до `getNumComponents` - 1.
- Первые `getNumColorComponents` индексы относятся к цветовым образцам.
- Если присутствует альфа-образец, он соответствует последнему индексу.
- Порядок цветовых индексов определяется `ColorSpace`. Обычно это отражает имя типа

цветового пространства (например, TYPE_RGB), индекс 0 соответствует красному, индекс 1 - зеленому, а индекс 2 - синему.

Преобразование значений пикселей в цветовые / альфа-компоненты для целей отображения или обработки представляет собой однозначное соответствие образцов компонентам. А `PackedColorModel`обычно используется с данными изображения, которые используют маски для определения упакованных образцов. Например, `PackedColorModel` может использоваться в сочетании с а `SinglePixelPackedSampleModel` для построения `BufferedImage`. Обычно маски, используемые `SampleModel` и `ColorModel`, будут одинаковыми. Однако, если они отличаются, интерпретация цвета пиксельных данных выполняется в соответствии с масками `ColorModel`.

Для всех объектов этого класса допустимо представление в одном пикселе, поскольку всегда можно представить значения пикселей, используемые с этим классом, в одном `int`. Поэтому методы, использующие это представление, не выбрасывают значение `IllegalArgumentException` из-за недопустимого значения пикселя.

Подклассом `PackedColorModel` являктся `DirectColorModel`

Класс DirectColorModel

Класс `DirectColorModel` – это класс `ColorModel`, который работает со значениями пикселей, которые представляют цветовую и альфа-информацию RGB *в виде отдельных выборок и которые упаковывают все выборки для одного пикселя в одно значение типа int, short или byte*. Этот класс может использоваться только с цветовыми пространствами типа ColorSpace.TYPE_RGB. Кроме того, для каждого компонента цветового пространства минимальное нормализованное значение компонента, полученное с помощью метода `getMinValue()` цветового пространства,

должно быть равно 0,0, а максимальное значение, полученное с помощью `getMaxValue()` метода, должно быть равно 1.0 (эти минимальные / максимальные значения типичны для пространств RGB). В значениях пикселей должно быть три цветовых образца, и может быть один альфа-образец. Для тех методов, которые используют примитивное представление типа в пикселях массива `transferType`, длина массива всегда равна единице. Поддерживаемые типы передачи - DataBuffer.TYPE_BYTE, DataBuffer.TYPE_USHORT и DataBuffer.ТИП_INT. Цветовые и альфа-образцы хранятся в одном элементе массива в битах, обозначенных битовыми масками. Каждая битовая маска должна быть смежной, и маски не должны перекрываться. Те же маски применяются к однопиксельному представлению int, используемому другими методами. Соответствие масок и образцов цвета / альфа выглядит следующим образом:

- Маски идентифицируются индексами, идущими от 0 до 2, если альфа отсутствует, или 3, если альфа присутствует.
- Первые три индекса относятся к цветовым образцам; индекс 0 соответствует красному, индекс 1 зеленому и индекс 2 синему.
- Индекс 3 соответствует альфа-образцу, если он присутствует.

Преобразование значений пикселей в цветовые / альфа-компоненты для целей отображения или обработки представляет собой однозначное соответствие образцов компонентам. Обычно `DirectColorModel` используется с данными изображения, которые используют маски для определения упакованных образцов. Например, `DirectColorModel`может быть использовано в сочетании с `SinglePixelPackedSampleModel` для построения a `BufferedImage`. Обычно маски, используемые the `SampleModel`и `ColorModel`, будут одинаковыми. Однако, если они отличаются,

интерпретация цвета пиксельных данных будет выполняться в соответствии с масками `ColorModel`.

Для всех объектов этого класса допустимо представление одного пикселя int, поскольку всегда можно представить значения пикселей, используемые с этим классом, в одном int. Следовательно, методы, которые используют это представление, не будут выбрасывать `IllegalArgumentException` из-за недопустимого значения пикселя.

Эта цветовая модель похожа на визуализацию X11 TrueColor. Цветовая модель RGB по умолчанию, указанная методом `getRGBdefault`, является `DirectColorModel` со следующими параметрами:

```
Количество битов: 32
Красная маска: 0x00ff0000
Зеленая маска: 0x0000ff00
Синяя маска: 0x000000ff
Альфа-маска: 0xff000000
Цветовое пространство: sRGB
isAlphaPremultiplied: False
Прозрачность: Transparency.TRANSLUCENT
тип ПЕРЕДАЧИ: DataBuffer.TYPE_INT
```

Многие методы в этом классе являются final. Это связано с тем, что базовый собственный графический код делает предположения о компоновке и работе этого класса, и эти предположения отражены в реализациях методов здесь, которые отмечены как final. Вы можете создать подкласс этого класса, но вы не можете переопределить или изменить поведение этих методов.

Классы – фильтры

Класс ImageFilter

Класс ImageFilter является реализацией интерфейса ImageConsumer. Несмотря на свое название, этот класс не производит никакой фильтрации, он передает изображение без изменений. Для преобразования изображения этот класс надо расширить, переопределив метод setPixels(). Результат преобразования необходимо передать потребителю, роль которого выполняет поле consumer этого класса.

Далее описаны несколько готовых расширений этого класса.

Класс AreaAveragingScaleFilter

Этот подкласс ImageFilter предназначен для масштабирования изображений с использованием алгоритма усреднения по площади, который дает более плавные результаты, чем алгоритм ближайшего соседа.

Этот класс расширяет базовый класс ImageFilter для масштабирования существующего изображения и предоставления источника для нового изображения, содержащего передискретизированное изображение. Пиксели в исходном изображении смешиваются для получения пикселей для изображения указанного размера. Процесс смешивания аналогичен масштабированию исходного изображения до размера, кратного целевому, с использованием репликации пикселей, а затем его обратному масштабированию до целевого размера путем простого усреднения всех пикселей в увеличенном изображении, которые попадают в данный пиксель целевого изображения. Если данные из источника не доставляются в порядке сверху вниз, фильтр вернется к простому поведению репликации пикселей и использует метод requestTopDownLeftRightResend() для лучшей повторной фильтрации пикселей в конце.

Класс предназначен для использования в сочетании с объектом FilteredImageSource для создания масштабированных версий существующих

изображений. Из-за зависимостей реализации могут быть различия в значениях пикселей изображения, отфильтрованного на разных платформах.

Конструктор и описание
AreaAveragingScaleFilter(int width, int height) Создает AreaAveragingScaleFilter, который масштабирует пиксели исходного изображения в соответствии с параметрами width и height. **Параметры:** width - целевая ширина для масштабирования изображения height - целевая высота для масштабирования изображения

Методы AreaAveragingScaleFilter

Модификатор и тип	Способ и описание
void	**setHints**(int hints) Определите, доставляются ли данные с необходимыми подсказками, чтобы алгоритм усреднения мог выполнять свою работу. Примечание: Этот метод предназначен для вызова ImageProducerтемImage, чьи пиксели фильтруются. Разработчики, использующие этот класс для фильтрации пикселей изображения, должны избегать прямого вызова этого метода, поскольку эта операция может помешать операции фильтрации. **Параметры:** hints - набор подсказок, которые использует ImageConsumer для обработки пикселей
void	**setPixels**(int x, int y, int w, int h, **ColorModel** model, byte[] pixels, int off, int scansize) Объедините компоненты для доставленных байтовых пикселей в массивы накопления и отправьте любые усредненные данные для полных строк пикселей. Если правильные подсказки не были указаны в вызове setHints, тогда передайте работу нашему суперклассу, который способен масштабировать пиксели независимо от подсказок доставки. Примечание: Этот метод предназначен для вызова ImageProducerтемImage, чьи пиксели фильтруются. Разработчикам, использующим этот класс для фильтрации пикселей изображения, следует избегать прямого вызова этого метода, поскольку эта операция может помешать операции фильтрации. **Параметры:**

	x - координата X верхнего левого угла области пикселей, которую нужно установить y - координата Y верхнего левого угла области пикселей, которую нужно установить w - ширина области пикселей h - высота области пикселей model - указанный ColorModel pixels - массив пикселей off - смещение в pixelsмассив scansize - расстояние от одного ряда пикселей до следующего в массиве pixels
void	**setPixels**(int x, int y, int w, int h, **ColorModel** model, int[] pixels, int off, int scansize) Объедините компоненты для доставленных int-пикселей в массивы накопления и отправьте любые усредненные данные для полных строк пикселей. Отличается от предыдущего метода только типом данных массива pixels

Класс BufferedImageFilter

Подкласс BufferedImageFilter класса ImageFilter обеспечивают средство использования оператора изображения с одним источником или одним назначением (BufferedImageOp) для фильтрации BufferedImage в парадигме производитель – потребитель - наблюдатель. Примерами этих операторов изображения являются: ConvolveOp, AffineTransformOp и LookupOp.

Конструктор и описание
BufferedImageFilter(**BufferedImageOp** op) Создает BufferedImageFilter с указанным оператором с одним источником или одним назначением. **Параметры:** op - указанный BufferedImageOp для использования для фильтрации BufferedImage **Выбрасывает:**

NullPointerException - если значение op равно нулю

Методы класса BufferedImageFilter

Модификатор и тип	Способ и описание
BufferedImageOp	getBufferedImageOp() Возвращает BufferedImageOp.
void	imageComplete(int status) Фильтрует информацию, предоставленную в imageCompleteметоде ImageConsumerинтерфейса.
void	setColorModel(ColorModel model) Фильтрует информацию, предоставленную в методе setColorModel интерфейса ImageConsumer.
void	setDimensions(int width, int height) Фильтрует информацию, предоставленную в методе setDimensions интерфейса ImageConsumer.
void	setPixels(int x, int y, int w, int h, ColorModel model, byte[] pixels, int off, int scansize) Фильтрует информацию, предоставленную в методе setPixels интерфейса ImageConsumer, который принимает массив байтов.
void	setPixels(int x, int y, int w, int h, ColorModel model, int[] pixels, int off, int scansize) Фильтрует информацию, предоставленную в setPixelsметоде ImageConsumerинтерфейса, который принимает массив целых чисел.

Класс CropImageFilter

Класс CropImageFilter предназначен для обрезки изображений. Этот класс расширяет базовый класс ImageFilter для извлечения заданной прямоугольной области существующего изображения и предоставления источника для нового изображения, содержащего только извлеченную область. Он предназначен для использования в сочетании с объектом

FilteredImageSource *для создания обрезанных версий существующих изображений.*

Конструктор и описание
CropImageFilter(int x, int y, int w, int h) Создает CropImageFilter, который извлекает прямоугольную область пикселей из исходного изображения, как указано параметрами x, y, w и h. **Параметры:** x - расположение x в верхней части прямоугольника, который нужно извлечь y - расположение y верхней части извлекаемого прямоугольника w - ширина извлекаемого прямоугольника h - высота извлекаемого прямоугольника

Методы CropImageFilter

Модификатор и тип	Метод и описание
void	**setDimensions**(int w, int h) Переопределяет размеры исходного изображения и передает размеры прямоугольной обрезанной области ImageConsumer. Примечание: Этот метод предназначен для вызова ImageProducerтемImage, чьи пиксели фильтруются. *Разработчики, использующие этот класс для фильтрации пикселей изображения, должны избегать прямого вызова этого метода, поскольку эта операция может помешать операции фильтрации.* **Параметры:** w - ширина исходного изображения h - высота исходного изображения
void	**setPixels**(int x, int y, int w, int h, **ColorModel** model, byte[] pixels, int off, int scansize) Определите, пересекают ли доставленные байтовые пиксели область, подлежащую извлечению, и проходят ли только через то подмножество пикселей, которые отображаются в выходной области. Примечание: Этот метод предназначен для вызова ImageProducer тем Image, чьи пиксели фильтруются. *Разработчики, использующие этот класс для фильтрации пикселей изображения, должны избегать прямого вызова этого метода, поскольку эта операция может помешать операции фильтрации.* **Параметры:** x - координата X верхнего левого угла области пикселей, которую нужно установить

	y - координата Y верхнего левого угла области пикселей, которую нужно установить w - ширина области пикселей h - высота области пикселей model - указанный ColorModel pixels - массив пикселей off - смещение в pixelsмассив scansize - расстояние от одного ряда пикселей до следующего в pixelsмассиве
void	**setPixels**(int x, int y, int w, int h, **ColorModel** model, int[] pixels, int off, int scansize) Определите, пересекают ли доставленные int-пиксели область, подлежащую извлечению, и проходят ли они только через то подмножество пикселей, которое появляется в выходной области. Аналогичен предыдущему методу. Отличается только типом элементов массива pixels.
void	**setProperties**(**Hashtable**<?,?> props) Передает свойства исходного объекта после добавления свойства, указывающего обрезанную область. Примечание: Этот метод предназначен для вызова ImageProducer тем Image, чьи пиксели фильтруются. *Разработчики, использующие этот класс для фильтрации пикселей изображения, должны избегать прямого вызова этого метода, поскольку эта операция может помешать операции фильтрации.* **Параметры:** props - свойства исходного объекта

Класс GrayFilter

Фильтр изображения, который "отключает" изображение, превращая его в изображение в оттенках серого и делая пиксели изображения ярче. Используется кнопками для создания изображения для отключенной кнопки.

Конструктор и описание
GrayFilter(boolean b, int p) Создает объект GrayFilter, который преобразует цветное изображение в изображение в оттенках серого. **Параметры:**

```
b - логическое значение -- true, если пиксели должны быть ярче

p - значение int в диапазоне 0..100, которое определяет процент
серого, где 100 - самый темный серый, а 0 - самый светлый
```

Класс PixelGrabber

С помощью класса PixelGrabber приложение можно преобразовать изображение класса Image или любой его фрагмент прямоугольной формы в массив пикселов. В дальнейшем такой массив может быть обработан. На основе обработанного массива нетрудно создать новое изображение.

Класс PixelGrabber реализует ImageConsumer, который может быть присоединен к объекту Image или ImageProducer для извлечения подмножества пикселей в этом изображении.

Пример:

```
public void handlesinglepixel(int x, int y, int pixel) {
      int alpha = (pixel >> 24) & 0xff;
      int red   = (pixel >> 16) & 0xff;
      int green = (pixel >>  8) & 0xff;
      int blue  = (pixel      ) & 0xff;
      // Deal with the pixel as necessary...
 }


 public void handlepixels(Image img, int x, int y, int w, int h)
{
      int[] pixels = new int[w * h];
      PixelGrabber pg = new PixelGrabber(img, x, y, w, h,
pixels, 0, w);
      try {
          pg.grabPixels();
      } catch (InterruptedException e) {
          System.err.println("interrupted waiting for pixels!");
          return;
      }
```

```
        if ((pg.getStatus() & ImageObserver.ABORT) != 0) {
            System.err.println("image fetch aborted or errored");
            return;
        }
        for (int j = 0; j < h; j++) {
            for (int i = 0; i < w; i++) {
                handlesinglepixel(x+i, y+j, pixels[j * w + i]);
            }
        }
    }
```

Конструкторы PixelGrabber

Конструктор и описание
PixelGrabber(**Image** img, int x, int y, int w, int h, boolean forceRGB) Создайте объект PixelGrabber для захвата прямоугольной части пикселей (x, y, w, h) из указанного изображения. Пиксели накапливаются в исходной цветовой модели, если для каждого вызова SetPixels используется одна и та же цветовая модель, в противном случае пиксели накапливаются в цветовой модели RGB по умолчанию. Если параметр forceRGB имеет значение true, то пиксели будут накапливаться в цветовой модели RGB по умолчанию в любом случае. PixelGrabber выделяет буфер для хранения пикселей в любом случае. Если `(w < 0)` или `(h < 0)`, тогда они будут по умолчанию использовать оставшуюся ширину и высоту исходных данных, когда эта информация будет доставлена. **Параметры:** `img` - изображение для извлечения данных изображения из `x` - координата x верхнего левого угла прямоугольника пикселей для извлечения из изображения относительно стандартного (немасштабированного) размера изображения `y` - координата y верхнего левого угла прямоугольника пикселей для извлечения из изображения `w` - ширина прямоугольника пикселей для извлечения `h` - высота прямоугольника пикселей для извлечения `forceRGB` - true, если пиксели всегда должны быть преобразованы в цветовую модель RGB по умолчанию
PixelGrabber(**Image** img, int x, int y, int w, int h, int[] pix, int off, int scansize) Создайте объект PixelGrabber для захвата прямоугольной секции пикселей (x, y, w, h) из указанного изображения в заданный массив. Пиксели сохраняются в массиве в цветовой модели RGB по умолчанию. Данные RGB для пикселей (i, j), где (i, j) находится внутри прямоугольника (x, y, w, h), хранятся в массиве по адресу `pix[(j - y) * scansize + (i - x) + off]`.

Параметры:

img - изображение для извлечения пикселей из

x - координата x верхнего левого угла прямоугольника пикселей для извлечения из изображения относительно стандартного (немасштабированного) размера изображения

y - координата y верхнего левого угла прямоугольника пикселей для извлечения из изображения

w - ширина прямоугольника пикселей для извлечения

h - высота прямоугольника пикселей для извлечения

pix - массив целых чисел, которые должны использоваться для хранения пикселей RGB, извлеченных из изображения

off - смещение в массив, в котором должен храниться первый пиксель

scansize - расстояние от одного ряда пикселей до следующего в массиве

PixelGrabber(**ImageProducer** ip, int x, int y, int w, int h, int[] pix, int off, int scansize)

Создайте объект PixelGrabber для захвата прямоугольной секции пикселей (x, y, w, h) из изображения, созданного указанным ImageProducer, в заданный массив.

Пиксели сохраняются в массиве в цветовой модели RGB по умолчанию. Данные RGB для пикселя (i, j), где (i, j) находится внутри прямоугольника (x, y, w, h) хранится в массиве в пикселях [(j - y) * scansize + (i - x) + off].

Параметры:

ip - тотImageProducer, который создает изображение, из которого извлекаются пиксели

x - координата x верхнего левого угла прямоугольника пикселей для извлечения из изображения относительно стандартного (немасштабированного) размера изображения

y - координата y верхнего левого угла прямоугольника пикселей для извлечения из изображения

w - ширина прямоугольника пикселей для извлечения

h - высота прямоугольника пикселей для извлечения

pix - массив целых чисел, которые должны использоваться для хранения пикселей RGB, извлеченных из изображения

off - смещение в массив, в котором должен храниться первый пиксель

scansize - расстояние от одного ряда пикселей до следующего в массиве

Методы PixelGrabber

Модификатор и тип	Способ и описание
void	**abortGrabbing**() Запрос к PixelGrabber прервать выборку изображения.

ColorModel	**getColorModel**() Получить цветовую модель для пикселей, хранящихся в массиве.
int	**getHeight**() Получить высоту пиксельного буфера (после настройки высоты изображения).
Object	**getPixels**() Получить пиксельный буфер. Если PixelGrabber не был сконструирован с явным буфером пикселей для хранения пикселей, то этот метод будет возвращать значение null до тех пор, пока не будут известны размер и формат данных изображения. Поскольку PixelGrabber может отказаться от накопления данных в цветовой модели RGB по умолчанию в любое время, если исходное изображение использует более одной цветовой модели для доставки данных, объект массива, возвращаемый этим методом, может меняться со временем, пока захват изображения не будет завершен. **ВОЗВРАТ:** либо массив байтов, либо массив int
int	**getStatus**() Возвращает статус пикселей.
int	**getWidth**() Получить ширину пиксельного буфера (после настройки на ширину изображения).
boolean	**grabPixels**() Запросить изображение или ImageProducer начать доставку пикселей и дождаться, пока будут доставлены все пиксели в интересующем прямоугольнике. **ВОЗВРАТ:** true, если пиксели были успешно захвачены, false при прерывании, ошибке или тайм-ауте **Выбрасывает:** InterruptedException - Другой поток прервал этот поток.
boolean	**grabPixels**(long ms) Запросить изображение или ImageProducer начать доставку пикселей и дождитесь доставки всех пикселей в интересующем прямоугольнике или пока не истечет указанное время ожидания. Этот метод ведет себя следующим образом, в зависимости от значения ms: • If ms == 0, ожидает, пока не будут доставлены все пиксели

	• Если`ms > 0`, ожидает, пока все пиксели не будут доставлены по истечении времени ожидания.
	• If `ms < 0`, возвращает`true`, если все пиксели захвачены, `false`в противном случае и не ожидает.
	
	Параметры:
	`ms - количество миллисекунд, в течение которых требуется ждать прибытия пикселей изображения до истечения времени ожидания`
	
	ВОЗВРАТ:
	`true, если пиксели были успешно захвачены, false при прерывании, ошибке или тайм-ауте`
	
	Выбрасывает:
	`InterruptedException - Другой поток прервал этот поток.`

`void`	**`imageComplete`**`(int status)` Метод imageComplete является частью API ImageConsumer, который этот класс должен реализовать для извлечения пикселей. Примечание: Этот метод предназначен для вызова ImageProducer изображения, пиксели которого захватываются. Разработчикам, использующим этот класс для извлечения пикселей из изображения, следует избегать прямого вызова этого метода, поскольку эта операция может привести к проблемам с извлечением запрошенных пикселей. **Параметры:** `status - статус загрузки изображения`
`void`	**`setColorModel`**`(`**`ColorModel`**` model)` Метод setColorModel является частью API ImageConsumer, который этот класс должен реализовать для извлечения пикселей.
`void`	**`setDimensions`**`(int width, int height)` Метод setDimensions является частью API ImageConsumer, который этот класс должен реализовать для извлечения пикселей.
`void`	**`setHints`**`(int hints)` Метод setHints является частью API ImageConsumer, который этот класс должен реализовать для извлечения пикселей.
`void`	**`setPixels`**`(int srcX, int srcY, int srcW, int srcH,` **`ColorModel`**` model, byte[] pixels, int srcOff, int srcScan)` Метод SetPixels является частью API ImageConsumer, который этот класс должен реализовать для извлечения пикселей.

void	**setPixels**(int srcX, int srcY, int srcW, int srcH, **ColorModel** model, int[] pixels, int srcOff, int srcScan) Метод SetPixels является частью API ImageConsumer, который этот класс должен реализовать для извлечения пикселей.
void	**setProperties**(**Hashtable**<?,?> props) Метод setProperties является частью API ImageConsumer, который этот класс должен реализовать для извлечения пикселей.
void	**startGrabbing**() Попросить PixelGrabber начать выборку пикселей.
int	**status**() Возвращает статус пикселей. Возвращаются флаги ImageObserver, представляющие доступную информацию о пикселях. Этот метод и `getStatus` имеет ту же реализацию, но `getStatus` является предпочтительным методом, поскольку он соответствует соглашению об именовании методов поиска информации в форме "getXXX". **ВОЗВРАТ:** побитовое ИЛИ всех соответствующих флагов ImageObserver

Использование PixelGrabber

Сначала вам нужно подготовить массив для хранения пикселов достаточного размера:

```
pix = new int[nImageWidth * nImageHeight];
```

Здесь nImageWidth - ширина изображения в пикселах, а nImageHeight - высота.

Далее нужно создать объект класса PixelGrabber. Можно использовать один из конструкторов PixelGrabber, например:

```
public PixelGrabber(
  Image img, int x, int y, int w, int h,
  int pix[],int off, int scansize);
```

Здесь параметр img задает исходное изображение. Параметры x, y, w и h определяют координаты прямоугольной области, на основе которой строится массив пикселов pix.

С помощью параметра off можно задать смещение в массиве, начиная с которого туда будут записаны данные первого пиксела вырезаемой области. Параметр scansize определяет расстояние между строками пикселов в массиве. Фактически это количество столбцов в массиве.

Имеется выражение, с помощью которого можно определить, как пиксел с заданными координатами (i, j), расположенный в области (x, y, w, h) отображается на соответствующий элемент массива pix. Индекс idx данной точки в массиве pix будет равен следующей величине:

```
idx = (j - y) * scansize + (i - x) + off;
```

После создания объекта класса PixelGrabber можно выполнить заполнение массива, для чего следует вызвать метод grabPixels:

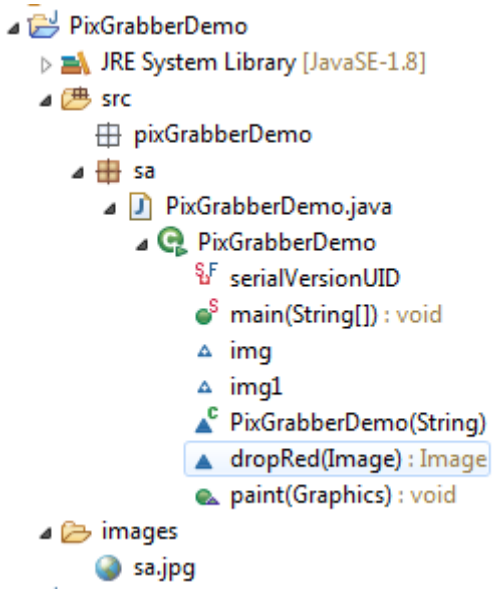
```
try
{
  pgr.grabPixels();
}
catch(InterruptedException ie)
{
}
```

Этот метод может создавать исключения InterruptedException.

Пример использования `PixelGrabber`

Пример демонстрирует как убрать красную составляющую цвета из изображения. Измененное изображение выводится.

Структура проекта



Класс PixGrabberDemo

```
package sa;

import java.awt.Graphics;
import java.awt.Image;
import java.awt.MediaTracker;
import java.awt.Toolkit;
import java.awt.image.MemoryImageSource;
import java.awt.image.PixelGrabber;

import javax.swing.JFrame;

public class PixGrabberDemo extends JFrame {
	private static final long serialVersionUID = 1L;
	Image img, img1;

	PixGrabberDemo(String s){
		super(s);
		//Запрашиваем исходное изображение
		Image img = Toolkit.getDefaultToolkit().getImage("images/sa.jpg");
		//Ждем пока изображение загрузится. Изучите класс MediaTracker!!!
		MediaTracker mt = new MediaTracker(this);
		mt.addImage(img, 0);
		try
		{
		  mt.waitForAll();
		}
		catch(InterruptedException ie)
		{
		  return;
		}
		//Определяем высоту и ширину загруженного изображения
		int h = img.getHeight(this);
		int w = img.getWidth(this);
		//Устанавливаем ширину и высоту окна как ширина и высота изображения
```

```
            this.setSize(w, h);
            //Определяем реакцию на закрытие окна
            this.setDefaultCloseOperation(EXIT_ON_CLOSE);
            //Вызываем метод, который удаляет из изображения красную составляющую
            //Метод описан ниже
            img1 = this.dropRed(img);
        }

        //Метод, удаляющий красную составляющую из изображения
        Image dropRed(Image img)
        {
          Image imgNew = null;
          int[] pix;

          // Определяем ширину и вычрту полученного в качестве пврвметра изображения
          int nImageWidth = img.getWidth(null);
          int nImageHeight = img.getHeight(null);
          //Создаем массив для хранения пикселей изображения
          pix = new int[nImageWidth * nImageHeight];
          //Создаем PixelGrabber
          PixelGrabber pgr = new PixelGrabber(img, 0, 0, nImageWidth, nImageHeight,
pix, 0, nImageWidth);
          //С помощью PixelGrabber забираем массив цветов точек изображения
          try
          {
            pgr.grabPixels();
          }
          catch(InterruptedException ie)
          {
            return null;
          }
          //Последовательно просматриваем массив цветов точек и
          //из каждого цвета удаляем красную составляющую
          for(int i = 0; i < pix.length; i++)
          {
            int nPixel = pix[i];
            int nG = 0xff & (nPixel >> 8);
            int nB = 0xff & nPixel;
            pix[i] = 0xff000000 | nG << 8 | nB;
          }
//Создаем новое изображение из измененного массива цветов с помощью MemoryImageSource
imgNew = this.createImage(new MemoryImageSource(nImageWidth, nImageHeight, pix, 0,
nImageWidth));
          return imgNew;
        }

        public void paint(Graphics g)
        {
              super.paint(g);
              //Выводим изображение без красной составляющей
              g.drawImage(img1, 0, 0, this);
        }

        public static void main(String[] args) {

              new PixGrabberDemo("PixGrabberDemo").setVisible(true);
        }

}
```

Класс ReplicateScaleFilter

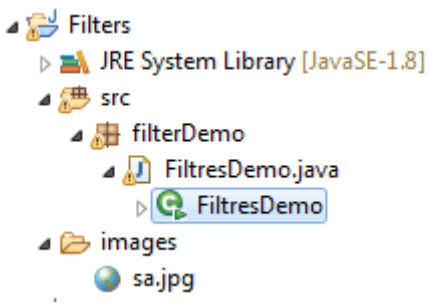
Класс ImageFilter для масштабирования изображений с использованием простейшего алгоритма. Этот класс расширяет базовый класс ImageFilter для масштабирования существующего изображения и предоставления источника для нового изображения, содержащего передискретизированное изображение. Пиксели в исходном изображении отбираются для получения пикселей для выходного изображения указанного размера путем репликации строк и столбцов пикселей для увеличения масштаба или пропуска строк и столбцов пикселей для уменьшения масштаба.

Он предназначен для использования в сочетании с объектом FilteredImageSource для создания масштабированных версий существующих изображений. Из-за зависимостей реализации могут быть различия в значениях пикселей изображения, отфильтрованного на разных платформах.

Конструктор и описание
ReplicateScaleFilter(int width, int height) Создает ReplicateScaleFilter, который масштабирует пиксели исходного изображения в соответствии с параметрами width и height. **Параметры:** width - целевая ширина для масштабирования изображения height - целевая высота для масштабирования изображения **Выбрасывает:** IllegalArgumentException - если width равно нулю или height равно нулю

Пример использования фильтров AreaAveragingScaleFilter, ReplicateScaleFilter, CropImageFilter

Структура проекта



Класс FiltresDemo

```
package filterDemo;

import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.image.AreaAveragingScaleFilter;
import java.awt.image.CropImageFilter;
import java.awt.image.FilteredImageSource;
import java.awt.image.ReplicateScaleFilter;
import javax.swing.JFrame;

public class FiltresDemo extends JFrame {
private static final long serialVersionUID = 1L;
private Image img, simg, croping, scroping, replimg, averimg;
//Конструктор
FiltresDemo(String s){
super(s);
//Получаем изображение из файла
img = Toolkit.getDefaultToolkit().getImage("images/sa.jpg");
//Создаем фильтры
//Фильтр для обрезки изображения
CropImageFilter cif = new CropImageFilter(200, 150, 600, 700);
//Простой фильтр для масштабирования в область 200 на 200 пикселей
ReplicateScaleFilter rsf = new ReplicateScaleFilter(200, 200);
//Более сложный фильтр для масштабирования в область 200 на 200 пикселей
AreaAveragingScaleFilter asf = new AreaAveragingScaleFilter(200, 200);

//Масштабированное AreaAveragingScaleFilter исходное изображение
simg = this.createImage(new FilteredImageSource(img.getSource(), asf));

//Обрезанное и масштабированное ReplicateScaleFilter изображение
croping = this.createImage(new FilteredImageSource(img.getSource(), cif));
scroping = this.createImage(new FilteredImageSource(croping.getSource(), rsf));

//Масштабированное ReplicateScaleFilter исходное изображение
replimg = this.createImage(new FilteredImageSource(img.getSource(), rsf));
//Масштабированное AreaAveragingScaleFilter исходное изображение
CropImageFilter cif1 = new CropImageFilter(800, 70, 300, 300);
averimg = this.createImage(new FilteredImageSource(img.getSource(), cif1));
averimg = this.createImage(new FilteredImageSource(averimg.getSource(), asf));

this.setSize(400, 400);
this.setVisible(true);
this.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
public void paint(Graphics g) {
      //Выводим отфилтрованные изображения в разные места окна
      g.drawImage(simg, 0, 0, this);
      g.drawImage(scroping, 200, 0, this);
      g.drawImage(replimg, 0, 200, this);
      g.drawImage(averimg, 200, 200, this);
      }
public static void main(String[] args) {
      new FiltresDemo("Фильтры");
      }
}
```

Класс RGBImageFilter

Этот класс предоставляет способ создания ImageFilter, который изменяет пиксели изображения в цветовой модели RGB по умолчанию. Он предназначен для использования в сочетании с объектом FilteredImageSource для создания отфильтрованных версий существующих изображений. Это абстрактный класс, который предоставляет вызовы, необходимые для передачи всех данных пикселей через один метод, который преобразует пиксели по одному в цветовую модель RGB по умолчанию, независимо от цветовой модели, используемой ImageProducer. *Единственный метод, который необходимо определить для создания пригодного для использования фильтра изображения, - это метод filterRGB*.

Пример определения фильтра, который меняет местами красный и синий компоненты изображения:

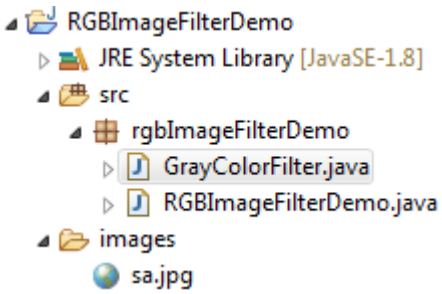
```
class RedBlueSwapFilter extends RGBImageFilter {
    public RedBlueSwapFilter() {
        canFilterIndexColorModel = true;
    }
    public int filterRGB(int x, int y, int rgb) {
        return ((rgb & 0xff00ff00)
                | ((rgb & 0xff0000) >> 16)
                | ((rgb & 0xff) << 16));
    }
}
```

Конструктор
RGBImageFilter()

Пример использования RGBImageFilter

Преобразование цветного изображения в изображение в градациях серого.

Структура проекта



Класс RGBImageFilterDemo

```
package rgbImageFilterDemo;

import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.image.FilteredImageSource;
import java.awt.image.RGBImageFilter;

import javax.swing.JFrame;

public class RGBImageFilterDemo extends JFrame{
	private static final long serialVersionUID = 1L;
	private Image img, newimg;
	//Конструктор
	public RGBImageFilterDemo(String s) {
		//Вызов конструктора суперкласса
		super(s);
		//Читаем изображение
		img = Toolkit.getDefaultToolkit().getImage("images/sa.jpg");
		//Создаем экземпляр RGBImageFilter типа GrayColorFilter
		RGBImageFilter rgb = new GrayColorFilter();
		//Создаем отфильтрованное изображние
		newimg = this.createImage(new FilteredImageSource(img.getSource(),rgb));
		this.setSize(800, 800);
		this.setDefaultCloseOperation(EXIT_ON_CLOSE);
		this.setVisible(true);
	}

	public void paint(Graphics g) {
		//Рисуем изображение
		g.drawImage(newimg, 0, 0, this);
	}

	public static void main(String[] args) {
		new RGBImageFilterDemo("RGBImageFilterDemo");
	}
}
```

Класс GrayColorFilter

```
package rgbImageFilterDemo;
```

```
import java.awt.image.RGBImageFilter;

public class GrayColorFilter extends RGBImageFilter {

	public GrayColorFilter() {
		this.canFilterIndexColorModel = true;
	}

	@Override
	public int filterRGB(int x, int y, int rgb) {
		//Получаем красную компоненту цвета
		int r = (0x000000ff & (rgb>>16));
		//Получаем зеленую компоненту цвета
		int g = (0x000000ff & (rgb>>8));
		//Получаем синюю компоненту цвета
		int b = (0x000000ff & rgb);
		//Вычисляем яркость точки
		int gray = (int)(0.56 * g + 0.33 * r + 0.11 * b);
		//Получаем цвет точки в оттенках серого такой же яркости, как яркость
исходной точки
		int grayColor = (int) (0xff000000 | gray << 16 | gray << 8 | gray);

		return grayColor;
	}

	}
```

Модель «Прямого доступа»

Основой модели прямого доступа является класс BufferedImage с доступныч будером данных изображения.

Класс BufferedImage

BufferedImage состоит из данных ColorModel (модель цвета) и Raster (растр изображения). Количество и типы полос в SampleModel (модель составляющих цвкта) Raster должны соответствовать количеству и типам, требуемым ColorModel для представления его цветовых и альфа-компонентов.

Все объекты BufferedImage имеют координату верхнего левого угла (0, 0). Поэтому любое Raster изображение, используемое для построения BufferedImage, должно иметь minX=0 и minY=0.

Поля BufferedImage

Модификатор и тип	Поле и описание
static int	**TYPE_3BYTE_BGR** Представляет изображение с 8-разрядными цветовыми компонентами RGB, соответствующими цветовой модели BGR в стиле Windows) с цветами Blue, Green и Red, сохраненными в 3 байтах.
static int	**TYPE_4BYTE_ABGR** Представляет изображение с 8-разрядными цветовыми компонентами RGBA с синими, зелеными и красными цветами, хранящимися в 3 байтах и 1 байте альфа. Представляет изображение с 8-разрядными цветовыми компонентами RGBA с синими, зелеными и красными цветами, хранящимися в 3 байтах и 1 байте альфа. Считается, что цветовые данные на этом изображении не должны быть предварительно умножены на альфа. Байтовые данные чередуются в однобайтовом массиве в порядке A, B, G, R от младших к старшим байтовым адресам в каждом пикселе.
static int	**TYPE_4BYTE_ABGR_PRE** Представляет изображение с 8-разрядными цветовыми компонентами RGBA с синими, зелеными и красными цветами, хранящимися в 3 байтах и 1 байте альфа. Цветовые данные в этом изображении считаются предварительно умноженными на альфа. Байтовые данные чередуются в однобайтовом массиве в порядке A, B, G, R от младших к старшим байтовым адресам внутри каждого пикселя.
static int	**TYPE_BYTE_BINARY** Представляет непрозрачное 1, 2 или 4-битное изображение с байтовой упаковкой. Изображение имеет символ `IndexColorModel` без альфа. Когда этот тип используется в качестве `imageType` аргумента конструктора `BufferedImage`, который принимает аргумент `imageType`аргумент, но не `ColorModel`аргумент, создается 1-разрядное изображение `IndexColorModel`с двумя цветами в sRGB по умолчанию `ColorSpace`: {0, 0, 0} и {255, 255, 255}. Изображения с 2 или 4 битами на пиксель могут быть созданы с помощью `BufferedImage`конструктора, который принимает `ColorModel`аргумент, предоставляя а `ColorModel`с соответствующим размером карты. Изображения с 8 битами на пиксель должны использовать типы изображений `TYPE_BYTE_INDEXED`или `TYPE_BYTE_GRAY`в зависимости от их `ColorModel`. Когда данные о цвете сохраняются в изображении этого типа, ближайший цвет в цветовой карте определяется с помощью `IndexColorModel`и сохраняется результирующий индекс. В зависимости от цветов в цветовой карте может произойти аппроксимация и потеря альфа- или цветовых компонентов`IndexColorModel`.

static int	**TYPE_BYTE_GRAY** Представляет изображение в оттенках серого в байтах без знака, неиндексированное.
static int	**TYPE_BYTE_INDEXED** Представляет индексированное байтовое изображение. Когда этот тип используется в качестве `imageType` аргумента конструктора `BufferedImage`, который принимает аргумент `imageType`, но не аргумент `ColorModel`, `IndexColorModel`создается с 256-цветной палитрой цветового куба 6/6/6 с остальными цветами из диапазона 216-255, заполненными значениями оттенков серого в цветовом пространстве sRGB по умолчанию. Когда данные о цвете сохраняются в изображении этого типа, ближайший цвет в цветовой карте определяется с помощью `IndexColorModel`и сохраняется результирующий индекс. В зависимости от цветов в цветовой карте может произойти аппроксимация и потеря альфа- или цветовых компонентов`IndexColorModel`.
static int	**TYPE_CUSTOM** Тип изображения не распознается, поэтому это должно быть настроенное изображение. Этот тип используется только в качестве возвращаемого значения для метода GetType().
static int	**TYPE_INT_ARGB** Представляет изображение с 8-битными цветовыми компонентами RGBA, упакованными в целочисленные пиксели. Представляет изображение с 8-битными цветовыми компонентами RGBA, упакованными в целочисленные пиксели. Изображение имеет символ `DirectColorModel` с альфой. Считается, что цветовые данные на этом изображении не должны быть предварительно умножены на альфа. Когда этот тип используется в качестве `imageType` аргумента `BufferedImage` конструктора, созданное изображение согласуется с изображениями , созданными в JDK1.1 и более ранних версиях.
static int	**TYPE_INT_ARGB_PRE** Представляет изображение с 8-битными цветовыми компонентами RGBA, упакованными в целочисленные пиксели. Изображение имеет `DirectColorModel` с альфой. Цветовые данные в этом изображении считаются предварительно умноженными на альфа.
static int	**TYPE_INT_BGR** Представляет изображение с 8-разрядными цветовыми компонентами RGB, соответствующими цветовой модели BGR в стиле Windows или Solaris, с синим, зеленым и красным цветами, упакованными в целочисленные пиксели. Представляет изображение с 8-разрядными цветовыми компонентами RGB, упакованными в целочисленные пиксели. Изображение имеет `DirectColorModel` без альфы. Когда данные с непрозрачным альфа-символом хранятся в изображении этого типа, данные о цвете

	должны быть приведены к форме без предварительного умножения, а альфа-символ отброшен, как описано в `AlphaComposite`.
`static int`	**`TYPE_INT_RGB`** Представляет изображение с 8-разрядными цветовыми компонентами RGB, упакованными в целочисленные пиксели.
`static int`	**`TYPE_USHORT_555_RGB`** Представляет изображение с 5-5-5 цветовыми компонентами RGB (5-битный красный, 5-битный зеленый, 5-битный синий) без альфа. Это изображение имеет `DirectColorModel`. Когда данные с непрозрачной альфа-версией хранятся в изображении этого типа, данные о цвете должны быть приведены к форме без предварительного умножения, а альфа-версия отброшена, как описано в `AlphaComposite`
`static int`	**`TYPE_USHORT_565_RGB`** Представляет изображение с 5-6-5 цветовыми компонентами RGB (5-битный красный, 6-битный зеленый, 5-битный синий) без альфа. Это изображение имеет `DirectColorModel`. Когда данные с непрозрачной альфа-версией хранятся в изображении этого типа, данные о цвете должны быть приведены к форме без предварительного умножения, а альфа-версия отброшена, как описано в `AlphaComposite`
`static int`	**`TYPE_USHORT_GRAY`** Представляет собой короткое изображение без знака в оттенках серого, неиндексированное). Это изображение имеет а `ComponentColorModel` с CS_GRAY `ColorSpace`. Когда данные с непрозрачной альфа-версией хранятся в изображении этого типа, данные о цвете должны быть приведены к форме без предварительного умножения, а альфа-версия отброшена, как описано в `AlphaComposit`

Методы BufferedImage

Модификатор и тип	Метод и описание
`void`	**`addTileObserver`**(**`TileObserver`** `to`) Добавляет наблюдателя плитки.
`void`	**`coerceData`**(`boolean isAlphaPremultiplied`) Принудительно приводит данные в соответствие с состоянием, указанным в переменной `isAlphaPremultiplied`.
`WritableRaster`	**`copyData`**(**`WritableRaster`** `outRaster`) Вычисляет произвольную прямоугольную область `BufferedImage` и копирует ее в указанную `WritableRaster`.
`Graphics2D`	**`createGraphics`**() Создает а `Graphics2D`, который можно использовать для рисования в `BufferedImage`.
`WritableRaster`	**`getAlphaRaster`**() Возвращает `WritableRaster`, представляющий альфа-канал для `BufferedImage` объектов с `ColorModel` объектами, которые поддерживают отдельный альфа-канал, такой как `ComponentColorModel`и `DirectColorModel`.

ColorModel	**getColorModel**() Возвращает цветовую модель изображения ColorModel.
Raster	**getData**() Возвращает изображение в виде растра.
Raster	**getData**(**Rectangle** rect) Вычисляет и возвращает область BufferedImage, ограниченную прямоуглдьником.
Graphics	**getGraphics**() Этот метод возвращает а **Graphics2D**; используется для обратной совместимости.
int	**getHeight**() Возвращает высоту BufferedImage.
int	**getHeight**(**ImageObserver** observer) Возвращает высоту BufferedImage.
int	**getMinTileX**() Возвращает минимальный индекс плитки в направлении x.
int	**getMinTileY**() Возвращает минимальный индекс плитки в направлении y.
int	**getMinX**() Возвращает минимальную координату x для этого BufferedImage.
int	**getMinY**() Возвращает минимальную координату y для этого BufferedImage.
int	**getNumXTiles**() Возвращает количество плиток в направлении x.
int	**getNumYTiles**() Возвращает количество плиток в направлении y.
Object	**getProperty**(**String** name) Возвращает свойство изображения по имени.
Object	**getProperty**(**String** name, **ImageObserver** observer) Возвращает свойство изображения по имени.
String[]	**getPropertyNames**() Возвращает массив имен, распознанных с помощью **getProperty(String)** или null, если имена свойств не распознаны.
WritableRaster	**getRaster**() Возвращает **WritableRaster**.
int	**getRGB**(int x, int y) Возвращает целочисленный пиксель в цветовой модели RGB по умолчанию (TYPE_INT_ARGB) и цветовом пространстве sRGB по умолчанию.
int[]	**getRGB**(int startX, int startY, int w, int h, int[] rgbArray, int offset, int scansize)

	Возвращает массив целых пикселей в цветовой модели RGB по умолчанию (TYPE_INT_ARGB) и цветовом пространстве sRGB по умолчанию из части данных изображения.
SampleModel	**getSampleModel**() Возвращает SampleModel, связанный с этим BufferedImage.
ImageProducer	**getSource**() Возвращает объект, который создает пиксели для изображения.
Vector<**RenderedImage**>	**getSources**() Возвращает **Vector** - список **RenderedImage** объектов, которые являются непосредственными источниками, а не источниками этих непосредственных источников, данных изображения для этого BufferedImage.
BufferedImage	**getSubimage**(int x, int y, int w, int h) Возвращает часть изображения, определенное указанной прямоугольной областью.
Raster	**getTile**(int tileX, int tileY) Возвращает tile (tileX,tileY) .
int	**getTileGridXOffset**() Возвращает смещение x сетки плиток относительно начала координат, например, координату x местоположения плитки (0, 0).
int	**getTileGridYOffset**() Возвращает смещение y сетки плиток относительно начала координат, например, координату y местоположения плитки (0, 0).
int	**getTileHeight**() Возвращает высоту плитки в пикселях.
int	**getTileWidth**() Возвращает ширину плитки в пикселях.
int	**getTransparency**() Возвращает прозрачность.
int	**getType**() Возвращает тип изображения.
int	**getWidth**() Возвращает ширину элемента BufferedImage.
int	**getWidth**(**ImageObserver** observer) Возвращает ширину элемента BufferedImage.
WritableRaster	**getWritableTile**(int tileX, int tileY) Проверяет плитку для записи.
Point[]	**getWritableTileIndices**() Возвращает массив **Point** объектов, указывающих, какие плитки извлекаются для записи.
boolean	**hasTileWriters**() Возвращает, извлечен ли какой-либо фрагмент для записи.
boolean	**isAlphaPremultiplied**()

	Возвращает, был ли альфа-код предварительно умножен или нет.
boolean	**isTileWritable**(int tileX, int tileY) Возвращает, извлечена ли плитка в данный момент для записи.
void	**releaseWritableTile**(int tileX, int tileY) Отказывается от разрешения на запись в плитку.
void	**removeTileObserver**(**TileObserver** to) Удаляет наблюдателя за плиткой.
void	**setData**(**Raster** r) Задает прямоугольную область изображения для содержимого указанного Raster r, которое, как предполагается, находится в том же координатном пространстве, что и BufferedImage.
void	**setRGB**(int x, int y, int rgb) Устанавливает в этом пикселе BufferedImage указанное значение RGB.
void	**setRGB**(int startX, int startY, int w, int h, int[] rgbArray, int offset, int scansize) Устанавливает массив пикселей в цветовой модели RGB по умолчанию (TYPE_INT_ARGB) и цветовом пространстве sRGB по умолчанию в часть данных изображения.
String	**toString**() Возвращает строковое представление BufferedImage объекта.

Конструкторы BufferedImage

Конструктор и описание
BufferedImage(**ColorModel** cm, **WritableRaster** raster, boolean isRasterPremultiplied, **Hashtable**<?,?> properties) Создает новый BufferedImage с заданным ColorModel и Raster.
BufferedImage(int width, int height, int imageType) Создает BufferedImage изображение одного из предопределенных типов изображений. Для изображения используется пространство sRGB по умолчанию. **Параметры:** width - ширина созданного изображения height - высота созданного изображения imageType - тип созданного изображения
BufferedImage(int width, int height, int imageType, **IndexColorModel** cm) . Создает BufferedImage изображение одного из предопределенных типов изображений: TYPE_BYTE_BINARY или TYPE_BYTE_INDEXED . Если тип изображения - TYPE_BYTE_BINARY, количество записей в цветовой модели используется для определения, должно ли изображение иметь 1, 2 или 4 бита на

пиксель. Если цветовая модель имеет 1 или 2 записи, изображение будет иметь 1 бит на пиксель. Если в нем 3 или 4 записи, изображение будет иметь 2 бита на пиксель. Если в нем содержится от 5 до 16 записей, изображение будет иметь 4 бита на пиксель. В противном случае будет выдано исключение IllegalArgumentException.

Параметры:

`width` - ширина созданного изображения

`height` - высота созданного изображения

`imageType` - тип созданного изображения

`cm` – `IndexColorModel` созданного изображения

Исключения:

`IllegalArgumentException` - если тип изображения не является `TYPE_BYTE_BINARY` или `TYPE_BYTE_INDEXED`, или если тип изображения является `TYPE_BYTE_BINARY`, а цветовая карта содержит более 16 записей.

В конструкторе `BufferedImage(int width, int height, int imageType)` задаются размеры изображения и одна из констант: TYPE_INT_RGB, TYPE_INT_ARGB, TYPE_INT_ARGB_PRE, TYPE_INT_BRG, TYPE_3BYTE_BRG, TYPE_4BYTE_ABRG, TYPE_4BYTE_ABRG_PRE, TYPE_BYTE_GRAY, TYPE_BYTE_BINARY, TYPE_BYTE_INDEXED, TYPE_USHORT_565_RGB, TYPE_USHORT_555_RGB, TYPE_USHORT_GRAY.

То есть, каждый пиксел может занимать или 4 байта (INT, 4BYTE), 3 байта (3BYTE), 2 байта(USHORT) или 1 байт(BYTE).

Может использоваться цветовая модель RGB или быть добавлена альфа-составляющая (ARGB), или задан другой порядок расположения цветовых составляющих (BRG), или определено изображение в градациях серого (GRAY).

Для создания объектов `BufferedImage` часто обращаются к методам createImage класса Component с приведением типа:

```
BufferedImage bim = (BufferedImage) createImage(width, heigth);
```

При этом экземпляр получает от компонента способ описания цветов пикселей, цыет фона и цвет рисования.

Расположение точек в изображении определяется классом Raster или его подклассом WritableRaster. Эти классы задают систему координат изображения, предоставляют доступ к отдельным пикселам с помощью методов getPixel(), позволяют выделять фрагметны изображения методами getPixels(). Класс WritableRaster короме этого позволяет изменять отдельные пикселы изображения методами setPixel() или целые фрагменты изображения методами setPixels() и setRect().

Начало системы координат изображения – левый верхний угол – имеет координаты (minX, minY), не обязательно равные 0.

При создании экземпляра класса BufferedImage автоматически создается связанный с ним экземпляр WritableRaster класса.

Точки изображения хранятся в буфере в одномерном или двумерном массиве. Работа с буфером осуществляется с помощью одного из классов DataBufferByte, DataBufferInt, DataBufferShort или DataBufferUShort в зависимости от длины двнных в буфере.

Общие свойства этих классов определены в абстрактном суперклассе DataBuffer. В нем определены типы данных, хранящихся в буфере: TYPE_BYTE, TYPE_USHORT, TYPE_INT, TYPE_UNDEFINED.

При создании экземпляра классов Raster или WritableRaster создается экземпляр соответствующего подкласса DataBuffer.

Чтобы абстагироваться от способа хранения точек изображения, Raster может обращаться не к буферу DataBuffer, а к подклассам абстрактного класса SampleModel (модель составляющих цвета), предоставляющим доступ не к байтам буфера, а составляющим (sample) цвета.

Существуютследующие подклассы SampleModel:

- ComponentSampleModel – каждая составляющая цвета хранится в отдельном элементе массива DataBuffer;

- BandedSampleModel – данные хранятся по составляющим, составляющие одного цвета хранятся обычно в одном массиве, а DataBuffer

содержит двумерный массив: по массиву для каждой составляющей; этот класс расширяет ComponentSampleModel;

- PixelInterlivedSampleModel – все составляющие одного цвета хранятся в соседних элементах единственного массива DataBuffer; этот класс расширяет ComponentSampleModel;

- MultiPixelPackedSampleModel – цвет каждого пиксела содержит только одну составляющую, которая упакована в один элемент DataBuffer;

- SinglePixelPackedSampleModel – все составляющие цвета каждого пиксела хранятся в одном элеменете массива DataBuffer.

Таким образом, существует возможность обрабатывать точки изображения, используя их координаты в методах классов Raster, WritableRaster или обращаться к составляющим цвета пикселей методами классов SampleModel. Если же необходимо работать с отдельными байтами, то можно использовать методы классов DataBuffer.

Создание изображения

Для создания изображения необходимо:

- создать объект класса BufferedImage

image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);

- вызвать у image метод getRaster() для получения WritableRaster

WritableRaster raster = image.getRaster();

- установить цвет каждого пикселя с помощью вызова метода raster.setPixel().

Следует помнить, что пикселю нельзя присвоить значение типа Color, поскольку необходимо знать способ указания цветов для конкретного тина изображения. Если изображение имеет тип TYPE_INT_ARGB, то каждый пиксель описывается 4 значениями. Эти значения необходимо передать в виде массива типа int из 4 элементов, например,

```
int[] color = {0, 0, 0, 255};

raster.setPixel(i, j, color);
```

Для установки значений прямоугольной области пикселей необходимо в параметрах метода raster.setPixels() указать расположение левого верхнего уга прямоугольника, его штртгу и высоту, а также массив, содержащий выборочные (sample) значения цветов этой группы пикселей. Если изображение имеет тип TYPE_INT_ARGB, то следует указать величины красной, зеленой, синей составляющих и альфа-значение для первого, потом теже данные для второго и т.д. пикселей:

```
int[] pixels = new int[4 * w * h];
pixels[0] = …//Значение красного цвета для первого пиксела
pixels[1] = …//Значение зеленого цвета для первого пиксела
pixels[2] = …//Значение синего цвета для первого пиксела
pixels[3] = …//Значение альфа-канала для первого пиксела
…
raster.setPixels(x, y, w, h, pixels);
```

Класс Raster

Класс, представляющий прямоугольный массив пикселей. Растр инкапсулирует буфер данных, в котором хранятся выборочные значения, и SampleModel, который описывает, как найти заданное выборочное значение в буфере данных.

Растр определяет значения для пикселей, занимающих определенную прямоугольную область плоскости, не обязательно включая (0, 0). Прямоугольник, известный как ограничивающий прямоугольник растра и доступный с помощью метода getBounds, определяется значениями minX, minY, width и height . Значения minX и minY определяют координату верхнего левого угла растра. Ссылки на пиксели за пределами ограничивающего прямоугольника могут привести к возникновению исключения или могут привести к ссылкам на непреднамеренные элементы связанного с растром буфера

данных. Ответственность за предотвращение доступа к таким пикселям лежит на пользователе.

SampleModel описывает, как образцы растра хранятся в элементах примитивного массива буфера данных. Выборки могут храниться по одной на элемент данных, как в модели с пересечением пикселей или в модели с полосой выборки, или упаковываться по нескольку в элемент, как в однопиксельной или многопиксельной модели. SampleModel также определяет, являются ли выборки расширенными по знаку, позволяя хранить неподписанные данные в подписанных типах данных Java, таких как byte, short и int.

Хотя растр может находиться в любом месте плоскости, SampleModel использует простую систему координат, которая начинается с (0, 0). Таким образом, растр содержит коэффициент преобразования, который позволяет отображать местоположения пикселей между системой координат растра и системой SampleModel. Перевод из системы координат SampleModel в систему координат растра может быть получен методами getSampleModelTranslateX и getSampleModelTranslateY .

Растр может совместно использовать буфер данных с другим растром либо путем явного построения, либо с помощью методов CreateChild и createTranslatedChild. Растры, созданные с помощью этих методов, могут возвращать ссылку на растр, из которого они были созданы с помощью метода getParent. Для растра, который не был создан с помощью вызова createTranslatedChild или CreateChild, getParent вернет значение null.

Метод createTranslatedChild возвращает новый растр, который использует все данные текущего растра, но занимает ограничивающий прямоугольник той же ширины и высоты, но с другой начальной точкой. Например, если родительский растр занимал область от (10, 10) до (100, 100), а преобразованный растр был определен как

начинающийся с (50, 50), то пиксель (20, 20) родительского и пиксель (60, 60) дочернего растра занимают одинаковуюрасположение в буфере данных, совместно используемом двумя растрами. В первом случае (-10, -10) следует добавить к координате пикселя, чтобы получить соответствующую координату SampleModel, а во втором случае следует добавить (-50, -50).

Преобразование между родительским и дочерним растром может быть определено путем вычитания значений дочерних sampleModelTranslateX и sampleModelTranslateY из значений родительского.

Метод CreateChild может использоваться для создания нового растра, занимающего только подмножество ограничивающего прямоугольника его родительского элемента (с той же или преобразованной системой координат) или с подмножеством полос его родительского элемента.

Все конструкторы защищены. Правильный способ создания растра - использовать один из статических методов создания, определенных в этом классе. Эти методы создают экземпляры растра, которые используют стандартные чередующиеся, группированные и упакованные модели выборки и которые могут обрабатываться более эффективно, чем растр, созданный путем объединения сгенерированной извне модели выборки и буфера данных.

Методы класса Raster

Модификатор и тип	Способ и описание
static **WritableRaster**	**createBandedRaster**(**DataBuffer** dataBuffer, int w, int h, int scanlineStride, int[] bankIndices, int[] bandOffsets, **Point** location) Создает растр на основе модели BandedSampleModel с указанным буфером данных, шириной, высотой, шагом строки развертки, индексами банка и смещениями полосы.

static **WritableRaster**	**createBandedRaster**(int dataType, int w, int h, int scanlineStride, int[] bankIndices, int[] bandOffsets, **Point** location) Создает растр на основе модели BandedSampleModel с указанным типом данных, шириной, высотой, шагом линии развертки, индексами банка и смещениями полосы.
static **WritableRaster**	**createBandedRaster**(int dataType, int w, int h, int bands, **Point** location) Создает растр на основе модели BandedSampleModel с указанным типом данных, шириной, высотой и количеством полос.
Raster	**createChild**(int parentX, int parentY, int width, int height, int childMinX, int childMinY, int[] bandList) Возвращает новый растр, который совместно использует весь или часть буфера данных этого растра.
WritableRaster	**createCompatibleWritableRaster**() Создайте совместимый записываемый растр того же размера, что и этот растр, с той же SampleModel и новым инициализированным буфером данных.
WritableRaster	**createCompatibleWritableRaster**(int w, int h) Создайте совместимый WritableRaster с указанным размером, новой SampleModel и новым инициализированным буфером данных.
WritableRaster	**createCompatibleWritableRaster**(int x, int y, int w, int h) Создайте совместимый записываемый растр с указанным местоположением (minX, minY) и размером (width, height), новой SampleModel и новым инициализированным буфером данных.
WritableRaster	**createCompatibleWritableRaster**(**Rectangle** rect) Создайте совместимый записываемый растр с местоположением (minX, minY) и размером (width, height), указанными в rect, новой SampleModel и новым инициализированным буфером данных.
static **WritableRaster**	**createInterleavedRaster**(**DataBuffer** dataBuffer, int w, int h, int scanlineStride, int pixelStride, int[] bandOffsets, **Point** location)

	Создает растр на основе модели PixelInterleavedSampleModel с указанным буфером данных, шириной, высотой, шагом линии развертки, шагом пикселя и смещениями полосы.
static **WritableRaster**	**createInterleavedRaster**(int dataType, int w, int h, int scanlineStride, int pixelStride, int[] bandOffsets, **Point** location) Создает растр на основе модели PixelInterleavedSampleModel с указанным типом данных, шириной, высотой, шагом линии развертки, шагом пикселя и смещениями полосы.
static **WritableRaster**	**createInterleavedRaster**(int dataType, int w, int h, int bands, **Point** location) Создает растр на основе модели PixelInterleavedSampleModel с указанным типом данных, шириной, высотой и количеством полос.
static **WritableRaster**	**createPackedRaster**(**DataBuffer** dataBuffer, int w, int h, int scanlineStride, int[] bandMasks, **Point** location) Создает растр на основе SinglePixelPackedSampleModel с указанным буфером данных, шириной, высотой, шагом линии развертки и масками полос.
static **WritableRaster**	**createPackedRaster**(**DataBuffer** dataBuffer, int w, int h, int bitsPerPixel, **Point** location) Создает растр на основе модели MultiPixelPackedSampleModel с указанным буфером данных, шириной, высотой и битами на пиксель.
static **WritableRaster**	**createPackedRaster**(int dataType, int w, int h, int[] bandMasks, **Point** location) Создает растр на основе SinglePixelPackedSampleModel с указанным типом данных, шириной, высотой и масками полос.
static **WritableRaster**	**createPackedRaster**(int dataType, int w, int h, int bands, int bitsPerBand, **Point** location) Создает растр на основе упакованной SampleModel с указанным типом данных, шириной, высотой, количеством полос и битами на полосу.
static **Raster**	**createRaster**(**SampleModel** sm, **DataBuffer** db, **Point** location) Создает растр с указанными SampleModel и DataBuffer.
Raster	**createTranslatedChild**(int childMinX, int childMinY)

	Создайте растр с тем же размером, SampleModel и DataBuffer, что и этот, но с другим расположением.
static **WritableRaster**	**createWritableRaster**(**SampleModel** sm, **DataBuffer** db, **Point** location) Создает записываемый растр с указанной SampleModel и буфером данных.
static **WritableRaster**	**createWritableRaster**(**SampleModel** sm, **Point** location) Создает записываемый файл с указанной SampleModel.
Rectangle	**getBounds**() Возвращает ограничивающий прямоугольник этого растра.
DataBuffer	**getDataBuffer**() Возвращает буфер данных, связанный с этим растром.
Object	**getDataElements**(int x, int y, int w, int h, **Object** outData) Возвращает пиксельные данные для указанного прямоугольника пикселей в примитивном массиве типа TransferType.
Object	**getDataElements**(int x, int y, **Object** outData) Возвращает данные для одного пикселя в примитивном массиве типа TransferType.
int	**getHeight**() Возвращает высоту растра в пикселях.
int	**getMinX**() Возвращает минимально допустимую координату X растра.
int	**getMinY**() Возвращает минимально допустимую координату Y растра.
int	**getNumBands**() Возвращает количество полос (выборок на пиксель) в этом растре.
int	**getNumDataElements**() Возвращает количество элементов данных, необходимое для передачи одного пикселя с помощью методов getDataElements и setDataElements.

Raster	**getParent**() Возвращает родительский растр (если таковой имеется) этого растра или значение null.
double[]	**getPixel**(int x, int y, double[] dArray) Возвращает образцы в массиве double для указанного пикселя.
float[]	**getPixel**(int x, int y, float[] fArray) Возвращает выборки в массиве с плавающей запятой для указанного пикселя.
int[]	**getPixel**(int x, int y, int[] iArray) Возвращает образцы в массиве int для указанного пикселя.
double[]	**getPixels**(int x, int y, int w, int h, double[] dArray) Возвращает двойной массив, содержащий все выборки для прямоугольника пикселей, по одной выборке на элемент массива.
float[]	**getPixels**(int x, int y, int w, int h, float[] fArray) Возвращает массив с плавающей запятой, содержащий все выборки для прямоугольника пикселей, по одной выборке на элемент массива.
int[]	**getPixels**(int x, int y, int w, int h, int[] iArray) Возвращает массив int, содержащий все выборки для прямоугольника пикселей, по одной выборке на элемент массива.
int	**getSample**(int x, int y, int b) Возвращает выборку в указанном диапазоне для пикселя, расположенного в точке (x,y), в виде int.
double	**getSampleDouble**(int x, int y, int b) Возвращает выборку в указанном диапазоне для пикселя, расположенного в точке (x,y), как двойную.
float	**getSampleFloat**(int x, int y, int b) Возвращает выборку в указанном диапазоне для пикселя, расположенного в точке (x,y), в виде значения с плавающей запятой.

SampleModel	getSampleModel() Возвращает SampleModel, который описывает расположение данных изображения.
int	getSampleModelTranslateX() Возвращает перевод X из системы координат SampleModel в систему координат растра.
int	getSampleModelTranslateY() Возвращает перевод Y из системы координат SampleModel в систему координат растра.
double[]	getSamples(int x, int y, int w, int h, int b, double[] dArray) Возвращает выборки для указанного диапазона для указанного прямоугольника пикселей в двойном массиве, по одной выборке на элемент массива.
float[]	getSamples(int x, int y, int w, int h, int b, float[] fArray) Возвращает выборки для указанного диапазона для указанного прямоугольника пикселей в массиве с плавающей запятой, по одной выборке на элемент массива.
int[]	getSamples(int x, int y, int w, int h, int b, int[] iArray) Возвращает выборки для указанного диапазона для указанного прямоугольника пикселей в массиве int, по одной выборке на элемент массива.
int	getTransferType() Возвращает TransferType, используемый для переноса пикселей с помощью методов getDataElements и setDataElements .
int	getWidth() Возвращает ширину растра в пикселях.

Класс WritableRaster

Этот класс расширяет возможности растра для обеспечения возможности записи в пикселях.

Конструкторы этого класса защищены. Чтобы создать экземпляр WritableRaster, используйте один из фабричных методов createWritableRaster в классе Raster.

Методы класса WritableRaster

Модификатор и тип	Способ и описание
WritableRaster	createWritableChild(int parentX, int parentY, int w, int h, int childMinX, int childMinY, int[] bandList) Возвращает новый WritableRaster, который использует весь или часть буфера данных этого WritableRaster.
WritableRaster	createWritableTranslatedChild(int childMinX, int childMinY) Создайте WritableRaster с тем же размером, SampleModel и DataBuffer, что и этот, но с другим расположением.
WritableRaster	getWritableParent() Возвращает родительский WritableRaster (если таковой имеется) этого WritableRaster, иначе null .
void	setDataElements(int x, int y, int w, int h, Object inData) Задает данные для прямоугольника пикселей из примитивного массива типа TransferType .
void	setDataElements(int x, int y, Object inData) Устанавливает данные для одного пикселя из примитивного массива типа TransferType .
void	setDataElements(int x, int y, Raster inRaster) Задает данные для прямоугольника пикселей из входного растра.
void	setPixel(int x, int y, double[] dArray) Устанавливает пиксель в буфере данных, используя двойной массив выборок для ввода.
void	setPixel(int x, int y, float[] fArray) Устанавливает пиксель в буфере данных, используя плавающий массив выборок для ввода.
void	setPixel(int x, int y, int[] iArray)

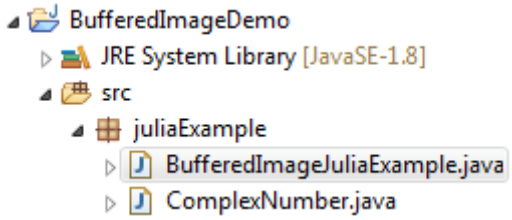
	Устанавливает пиксель в буфере данных, используя массив выборок int для ввода.
void	**setPixels**(int x, int y, int w, int h, double[] dArray) Устанавливает все выборки для прямоугольника пикселей из двойного массива, содержащего по одной выборке на элемент массива.
void	**setPixels**(int x, int y, int w, int h, float[] fArray) Устанавливает все выборки для прямоугольника пикселей из массива с плавающей запятой, содержащего по одной выборке на элемент массива.
void	**setPixels**(int x, int y, int w, int h, int[] iArray) Устанавливает все выборки для прямоугольника пикселей из массива int, содержащего по одной выборке на элемент массива.
void	**setRect**(int dx, int dy, **Raster** srcRaster) Копирует пиксели из растрового srcRaster в этот WritableRaster.
void	**setRect**(**Raster** srcRaster) Копирует пиксели из растрового srcRaster в этот WritableRaster.
void	**setSample**(int x, int y, int b, double s) Задает выборку в указанном диапазоне для пикселя, расположенного в точке (x, y) в буфере данных, используя double для ввода.
void	**setSample**(int x, int y, int b, float s) Задает выборку в указанном диапазоне для пикселя, расположенного в (x, y) в буфере данных, используя значение с плавающей точкой для ввода.
void	**setSample**(int x, int y, int b, int s) Задает выборку в указанном диапазоне для пикселя, расположенного в (x, y) в буфере данных, используя int для ввода.
void	**setSamples**(int x, int y, int w, int h, int b, double[] dArray) Задает выборки в указанном диапазоне для указанного прямоугольника пикселей из двойного массива, содержащего по одной выборке на элемент массива.
void	**setSamples**(int x, int y, int w, int h, int b, float[] fArray) Устанавливает выборки в указанном диапазоне для указанного прямоугольника пикселей из массива с плавающей запятой, содержащего по одной выборке на элемент массива.

void	**setSamples**(int x, int y, int w, int h, int b, int[] iArray) Устанавливает выборки в указанном диапазоне для указанного прямоугольника пикселей из массива int, содержащего по одной выборке на элемент массива.

Пример 1 создания изображения

В примере строится множество Жулиа.

Структура проекта



Класс BufferedImageJuliaExample

```
package juliaExample;

import java.awt.image.BufferedImage;
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JFrame;

public class BufferedImageJuliaExample {
    //Попробуйте разные варианты
    private static ComplexNumber c = new ComplexNumber(-0.223, 0.745);
    //private static ComplexNumber c = new ComplexNumber(-0.123, 0.745);
    //private static ComplexNumber c = new ComplexNumber(0.223, 0.1);

    //Массив пикселей для хранения фрактала
    private boolean[][] values = null;
    //Границы комплексной плоскости
    private double minX = -1.5;
    private double maxX = 1.5;
    private double minY = -1.5;
    private double maxY = 1.5;
    // BufferedImage для рисования фрактала
    private BufferedImage image = null;
    //Максимальное разрешенное значение ComplexNumer
    private double threshold = 1;
    //Глубина рекурсии
    private int iterations = 100;
    public BufferedImageJuliaExample(){
        // Создание объекта BufferedImage
        image = new BufferedImage(350,350,BufferedImage.TYPE_INT_RGB);
        // Заполняем booean[][]
        this.getValues();
```

```
        for(int i=0;i<350;i++){
            for(int j=0;j<350;j++){
            //Если точка принадлежит множеству Жулиа - она красная, если нет - желтая
             if(values[i][j]) image.setRGB(i, j, Color.RED.getRGB());
                if(!values[i][j]) image.setRGB(i, j, Color.YELLOW.getRGB());
            }
        }
        JFrame f = new JFrame("Множество Жулиа"){
                    private static final long serialVersionUID = 1L;
                    @Override
            public void paint(Graphics g){
                g.drawImage(image,0,0,null);
            }
        };
        f.setResizable(false);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(350,350);
        f.repaint();
        f.setVisible(true);
    }
    //Заполнение boolean[][] данными
    private void getValues(){
        values = new boolean[350][350];
        for(int i=0;i<350;i++){
            for(int j=0;j<350;j++){
                        double a = (double)i*(maxX-minX)/(double)350 + minX;
                        double b = (double)j*(maxY-minY)/(double)350 + minY;
                        values[i][j] = isInSet(new ComplexNumber(a,b));
            }
        }
    }
    private boolean isInSet(ComplexNumber cn){
        for(int i=0;i<iterations;i++){
            cn = cn.square().add(c);
        }
        return cn.magnitude()<Math.pow(threshold, 2);
    }

    public static void main(String[] args){
        new BufferedImageJuliaExample();
    }
      }
```

Класс ComplexNumber

```
package juliaExample;
public class ComplexNumber{
    private double a, b;
    // Создание комплексного числа из двух вещественных чисел
    public ComplexNumber(double a, double b){
        this.a = a;
        this.b = b;
    }
    // Возведение комплексных чисел в квадрат
    public ComplexNumber square(){
        return new ComplexNumber(this.a*this.a - this.b*this.b, 2*this.a*this.b);
    }
    // Сложение комплексных чисел
    public ComplexNumber add(ComplexNumber cn){
        return new ComplexNumber(this.a+cn.a, this.b+cn.b);
    }
```

```
    // Вычисление модуля^2
    public double magnitude(){
        return a*a+b*b;
    }
      }
```

Пример 2 создания изображения

В примере строится множество Мандельброта.

Демонстрируется использование класса ColorModel для преобразования значений цвета Color в выборочные данные пикселей. *Измените тип цвета для изображения на TYPE_BYTE_GRAY.*

```
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.awt.image.ColorModel;
import java.awt.image.WritableRaster;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class BufferedImageMandelbrotExample {

	public static void main(String[] args) {
		JFrame f = new RasterImageFrame();
		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		f.setVisible(true);
	}
}

class RasterImageFrame extends JFrame {
	private static final long serialVersionUID = 1L;
public RasterImageFrame() {
	super("Множество Мандельброта");
	this.setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
	BufferedImage image = makeMandebrot(DEFAULT_WIDTH, DEFAULT_HEIGHT);
	this.add(new JLabel(new ImageIcon(image)));
}

public BufferedImage makeMandebrot(int width, int height) {
	BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_ARGB);
	WritableRaster raster = image.getRaster();
	ColorModel model = image.getColorModel();
	Color fractalColor = Color.RED;
	int argb = fractalColor.getRGB();
	Object colorData = model.getDataElements(argb, null);
	for (int i = 0; i < width; i++)
		for (int j = 0; j < height; j++) {
			double a = XMIN + i * (XMAX - XMIN)/width;
			double b = YMIN + j * (YMAX - YMIN)/height;
			if (!escapesToInfinity(a, b))
				raster.setDataElements(i, j, colorData);
		}
	return image;
}

private boolean escapesToInfinity(double a, double b) {
```

```
        double x = 0.0;
        double y = 0.0;
        int iterations = 0;
        while ((x <= 2) && (y <= 2) && (iterations < MAX_ITERATIONS)) {
                double xnew = x * x - y * y + a;
                double ynew = 2 * x * y + b;
                x = xnew;
                y = ynew;
                iterations++;
        }
        return x > 2 || y > 2;
}
private static final double XMIN = -2;
private static final double XMAX = 2;
private static final double YMIN = -2;
private static final double YMAX = 2;
private static final int MAX_ITERATIONS = 20;
private static final int DEFAULT_WIDTH = 400;
private static final int DEFAULT_HEIGHT = 400;
        }
```

Чтение и запись изображений

Основным для Java Image I/O API является класс ImageIO пакета javax.imageio. Этот класс, содержит статические методы для чтения и записи файлов.

Класс ImageIO выбирает соответствующую программу *чтения* на основе типа файла. Он проверяет расширение файла и соответствующее значение в заголовке файла. Если для чтения файла нельзя найти подходящей программы или она не может расшифровать содержимое, то статический метод read() возвращает null.

Запись изображения в файл осуществляется с помощью метода write().

Для выполнения операций записи и чтения, которые выходят за пределы использования методов read() и write() необходимо получить соответствующие объекты классов ImageReader и ImageWrtter.

ImageReader - абстрактный суперкласс для синтаксического анализа и декодирования изображений. Этот класс является предком классов, которые читают в изображениях в контексте Java Image I / O framework. Объекты ImageReader обычно создаются классом service provider interface (SPI) для определенного формата. Классы поставщика услуг (например, экземпляры ImageReaderSpi) регистрируются в IIORegistry, который

использует их для распознавания форматов и представления доступных средств чтения и записи форматов.

ImageWriter - абстрактный суперкласс для кодирования и записи изображений. ImageWriter объекты обычно создаются классом поставщика услуг для определенного формата. Классы поставщика услуг зарегистрированы в IIORegistry, который использует их для распознавания форматов и представления доступных средств чтения и записи форматов.

Некоторые графические файлы, например, анимированные GIF, могут содержать несколько изображений. Однако метод read() класса ImageIO позволяет считывать только одно из них. Для чтения нескольких изображений необходимо преобразовать источник ввода данных в объект *ImageInputStream*.

Класс *ImageOutputStream* выполняет аналогичные функции для записи изображений.

Класс ImageIO

Методы Класс ImageIO

`static` **`ImageInputStream`**	**`createImageInputStream`**`(`**`Object`** `input)` Возвращает объект `ImageInputStream`, который будет принимать входные данные из заданного `Object`. Запрашивается набор `ImageInputStreamSpi`, зарегистрированных в классе `IIORegistry`, и первый, который способен принимать входные данные от предоставленного объекта , используется для создания возвращаемого `ImageInputStream`. Если подходящего `ImageInputStreamSpi` не существует, то возвращается `null`. Текущие настройки кэша из `getUseCache` и `getCacheDirectory` будут использоваться для управления кэшированием. **Параметры:** `input - элемент Object, который будет использоваться в качестве источника ввода, например, File, readable RandomAccessFile или InputStream.`

ВОЗВРАТ:
ImageInputStream, или null.

Выдает:
IllegalArgumentException - если input - null.

IOException - если файл кэша необходим, но не может быть создан.

static **ImageOutputStream** **createImageOutputStream**(**Object** output)

Возвращает объект ImageOutputStream, который будет отправлять свои выходные данные в данный Object.

Запрашивается набор ImageOutputStreamSpi, зарегистрированных в классе IIORegistry, и первый, который кто способен отправлять выходные данные из предоставленного объекта, используется для создания возвращаемого ImageOutputStream. Если подходящего ImageOutputStreamSpi не существует, возвращается null.

Текущие настройки кэша из getUseCache и getCacheDirectory будут использоваться для управления кэшированием.

Параметры:

output - Object, который будет использоваться в качестве назначения вывода, например, File, доступный для записи, RandomAccessFile или OutputStream.

ВОЗВРАТ:
ImageOutputStream, или null.

Выдает:
IllegalArgumentException - если output - null.

static **File** **getCacheDirectory**()

Возвращает текущее значение, установленное с помощью setCacheDirectory или null, если явная настройка не была выполнена.

ВОЗВРАТ:
File - указание каталога, в котором будут создаваться файлы кэша, или null, как указание системного каталога временных файлов по умолчанию.

static **ImageReader**	**getImageReader**(**ImageWriter** writer) Возвращает значение ImageReader, соответствующее данному ImageWriter, если таковое имеется, или null если подключаемый модуль для этого ImageWriterне указывает соответствующее ImageReader, или если данное ImageWriter значение не зарегистрировано. Этот метод предусмотрен главным образом для симметрии с getImageWriter(ImageReader). Обратите внимание, что этот метод возвращает "предпочтительный" читатель, который является первым в списке, возвращаемом javax.imageio.spi.ImageWriterSpi..getImageReaderSpiNames() **Параметры:** writer - экземпляр зарегистрированного ImageWriter. **ВОЗВРАТ:** ImageReader, или null. **Выдает:** IllegalArgumentException - если writer - null.
static **Iterator**<**ImageReader**>	**getImageReaders**(**Object** input) Возвращает Iterator, содержащий все зарегистрированные ImageReader в данный момент, которые могут декодировать предоставленные Object, обычно ImageInputStream. Положение потока остается в его предыдущем положении при выходе из этого метода. **Параметры:** input - ImageInputStream или другой Object, содержащее закодированные данные изображения. **ВОЗВРАТ:** Iterator, содержащий ImageReaders. **Выдает:** IllegalArgumentException - если inputесть null.
static **Iterator**<**ImageReader**>	**getImageReadersByFormatName**(**String** formatName)

	Возвращает Iterator, содержащий все зарегистрированные ImageReader которые декодировать именованный формат. **Параметры:** formatName - String, содержащая неофициальное название формата (*например*, "jpeg" или "tiff". **ВОЗВРАТ:** Iterator, содержащий ImageReaders. **Выдает:** IllegalArgumentException - если formatName - null.
static Iterator<ImageReader>	**getImageReadersByMIMEType**(String MIMEType) Возвращает Iterator, содержащий все зарегистрированные ImageReader, которые утверждают, что могут декодировать файлы с заданным типом MIME. **Параметры:** MIMEType - String, содержащий суффикс файла (*например*, "изображение /jpeg" или "изображение /x-bmp"). **ВОЗВРАТ:** Iterator, содержащий ImageReaders. **Выдает:** IllegalArgumentException - если MIMEType - null.
static Iterator<ImageReader>	**getImageReadersBySuffix**(String fileSuffix) Возвращает Iterator, содержащий все зарегистрированные ImageReader, которые утверждают, что могут декодировать файлы с заданным суффиксом. **Параметры:** fileSuffix - String, содержащая суффикс файла (*например*, "jpg" или "tiff"). **ВОЗВРАТ:** Iterator, содержащий ImageReaders. **Выдает:** IllegalArgumentException - если fileSuffix - null.
static Iterator<ImageTranscoder>	**getImageTranscoders**(ImageReader reader, ImageWriter writer)

Возвращает Iterator, содержащий все зарегистрированные ImageTranscoder которые утверждают, что могут перекодировать метаданные данного ImageReader и ImageWriter.

Параметры:

reader - ImageReader.

writer - ImageWriter.

ВОЗВРАТ:
Iterator, содержащий ImageTranscoder.

Выдает:
IllegalArgumentException - если reader или writer - null.

static **ImageWriter**

getImageWriter(**ImageReader** reader)

Возвращает значение ImageWriter, соответствующее данномуImageReader, если оно есть, или nullесли подключаемый модуль для этого ImageReader не указывает соответствующее ImageWriter, или если данное значение ImageReader не зарегистрировано.

Этот механизм может быть использован для получения anImageWriter, который будет понимать внутреннюю структуру непиксельных метаданных (как закодированных IIOMetadataобъектами), генерируемых ImageReader. Получив эти данные отImageReader и передавая его ImageWriterполученному с помощью этого метода, клиентская программа может прочитать изображение, каким-то образом изменить его и записать обратно, сохранив все метаданные, без необходимости понимать что-либо о структуре метаданных или даже о формате изображения. Обратите внимание, что этот метод возвращает "предпочтительный" элемент записи, который является первым в списке, возвращаемом javax.imageio.spi.ImageReaderSpi.getImageWriterSpiNames().

Параметры:

reader - экземпляр зарегистрированного ImageReader.

ВОЗВРАТ:
ImageWriter, или null.

Выбрасывает:
IllegalArgumentException - если reader - null.

static **Iterator**<**ImageWriter**>	**getImageWriters**(**ImageTypeSpecifier** type, **String** formatName) Возвращает Iterator, содержащий все зарегистрированные ImageWriter, которые утверждают, что могут кодировать изображения данного макета (заданного с помощью ImageTypeSpecifier) в заданном формате. **Параметры:** type – ImageTypeSpecifier указывает расположение изображения, которое должно быть записано. formatName - неофициальное название format. **ВОЗВРАТ:** Iterator, содержащий ImageWriters. **Выбрасывает:** IllegalArgumentException - если какой-либо параметр равен null.
static **Iterator**<**ImageWriter**>	**getImageWritersByFormatName**(**String** formatName) Возвращает Iterator, содержащий все зарегистрированные ImageWriter, которые утверждают, что могут кодировать именованный формат. **ВОЗВРАТ:** массив String.
static **Iterator**<**ImageWriter**>	**getImageWritersByMIMEType**(**String** MIMEType) Возвращает Iterator, содержащий все зарегистрированные ImageWriter, которые утверждают, что могут кодировать файлы с заданным типом MIME. **Параметры:** MIMEType - aString, содержащий суффикс файла (*например*, "image /jpeg" или "image /x-bmp"). **ВОЗВРАТ:** anIterator, содержащий ImageWriter. **Выдает:** IllegalArgumentException - если MIMEType - null.
static **Iterator**<**ImageWriter**>	**getImageWritersBySuffix**(**String** fileSuffix) Возвращает Iterator, содержащий все зарегистрированные ImageWriter, которые

	утверждают, что могут кодировать файлы с заданным суффиксом. **Параметры:** fileSuffix - aString, содержащий суффикс файла (*например*, "jpg" или "tiff"). **ВОЗВРАТ:** Iterator, содержащий ImageWriter. **Выбрасывает:** IllegalArgumentException - если fileSuffix - null.
static String[]	**getReaderFileSuffixes**() Возвращает массив Strings, в котором перечислены все файловые суффиксы, связанные с форматами, понятными текущему набору зарегистрированных пользователей. **ВОЗВРАТ:** массив String.
static String[]	**getReaderFormatNames**() Возвращает массив Strings, в котором перечислены все неофициальные имена форматов, понятные текущему набору зарегистрированных читателей. ВОЗВРАТ: массив String.
static String[]	**getReaderMIMETypes**() Возвращает массив String, в котором перечислены все типы MIME, понятные текущему набору зарегистрированных читателей. **ВОЗВРАТ:** массив String.
static boolean	**getUseCache**() Возвращает текущее значение, установленное с помощью setUseCache или true, если явная настройка не была выполнена. **Возвращает:** true, если для ImageInputStream может использоваться дисковый кэш ImageOutputStream.
static String[]	**getWriterFileSuffixes**() Возвращает массив Strings, в котором перечислены все файловые суффиксы, связанные с форматами, понятными текущему набору зарегистрированных писателей.

ВОЗВРАТ:
массив String.

static **String**[]	**getWriterFormatNames**() Возвращает массив Strings, в котором перечислены все неофициальные имена форматов, понятные текущему набору зарегистрированных писателей. **Параметры:** formatName - aString, содержащий неофициальное название формата (*например*, "jpeg" или "tiff". **ВОЗВРАТ:** anIterator, содержащий ImageWriter. **Выдает:** IllegalArgumentException - если formatNameесть null.
static **String**[]	**getWriterMIMETypes**() Возвращает массив Strings, в котором перечислены все типы MIME, понятные текущему набору зарегистрированных писателей. **ВОЗВРАТ:** массив String.
static **BufferedImage**	**read**(**File** input) Возвращает a BufferedImageкак результат декодирования a, поставляемого Fileс ImageReaderавтоматически выбранным из числа зарегистрированных в настоящее время. Текущие настройки кэша из getUseCache и getCacheDirectory будут использоваться для управления кэшированием в ImageInputStream. Обратите внимание, что не существует read - метода, который принимает имя файла в качестве String; используйте этот метод после создания File из имени файла. **Параметры:** input - File для чтения. **ВОЗВРАТ:** BufferedImage, содержащий декодированное содержимое входных данных, или null. **Выдает:**

	IllegalArgumentException - если input - null. IOException - если во время чтения возникает ошибка.
static BufferedImage	**read**(**ImageInputStream** stream) Возвращает a BufferedImageкак результат декодирования, поставляемого ImageInputStreamс ImageReader автоматически выбранным из числа зарегистрированных в данный момент. В отличие от большинства других методов в этом классе, этот метод закрывает предоставленный ImageInputStream после завершения операции чтения, если null не возвращается, то в этом случае этот метод *не* закрывает поток. **Параметры:** stream – ImageInputStream элемент для чтения. **ВОЗВРАТ:** BufferedImage, содержащий декодированное содержимое входных данных, или null. **Выдает:** IllegalArgumentException - если streamесть null. IOException - если во время чтения возникает ошибка.
static BufferedImage	**read**(**InputStream** input) Возвращает a BufferedImageкак результат декодирования а, поставляемого InputStreamс ImageReaderавтомати чески выбранным из числа зарегистрированных в настоящее время. Текущие настройки кэша из getUseCache и getCacheDirectory будут использоваться для управления кэшированием в создаваемом ImageInputStream. Этот метод *не* закрывает предоставленный InputStream после завершения операции чтения; вызывающий объект обязан закрыть поток. **Параметры:**

	input - InputStream для чтения. **ВОЗВРАТ:** BufferedImage, содержащий декодированное содержимое входных данных, или null. **Выдает:** IllegalArgumentException - если inputесть null. IOException - если во время чтения возникает ошибка.
static **BufferedImage**	**read**(**URL** input) Возвращает a BufferedImageкак результат декодирования a, поставляемого URLс ImageReaderавтоматически выбранным из числа зарегистрированных в настоящее время. Текущие настройки кэша из getUseCache и getCacheDirectory будут использоваться для управления кэшированием создаваемом в ImageInputStream. Этот метод не пытается найти ImageReader, которые могут считываться непосредственно из URL; это может быть выполнено с помощью IIORegistry и ImageReaderSpi. **Параметры:** input - URL для чтения. **ВОЗВРАТ:** BufferedImage, содержащий декодированное содержимое входных данных, или null. **Выдает:** IllegalArgumentException - если input - null. IOException - если во время чтения возникает ошибка.
static void	**scanForPlugins**() Сканирует плагины в пути к классу приложения, загружает их классы поставщика услуг и регистрирует экземпляр поставщика услуг для каждого найденного с помощью IIORegistry. Этот метод необходим, поскольку теоретически путь к классу приложения может измениться или могут стать доступными дополнительные плагины. Вместо повторного сканирования пути к классу при каждом

	вызове API, путь к классу сканируется автоматически только при первом вызове. Клиенты могут вызвать этот метод, чтобы запросить повторное сканирование. Таким образом, этот метод должен вызываться только сложными приложениями, которые динамически делают новые плагины доступными во время выполнения. Метод `getResources` контекста `ClassLoader` используется для поиска файлов JAR, содержащих файлы с именем `META-INF/services/javax.imageio.spi.`*classname*, где *classname* является одним из `ImageReaderSpi`, `ImageWriterSpi`, `ImageTranscoderSpi`, `ImageInputStreamSpi`, или `ImageOutputStreamSpi`, по пути к классу приложения. Содержимое найденных файлов указывает имена реальных классов реализации, которые реализуют вышеупомянутые интерфейсы поставщика услуг; затем загрузчик классов по умолчанию используется для загрузки каждого из этих классов и создания экземпляра каждого класса, который затем помещается в реестр для последующего извлечения. Точный набор искомых местоположений зависит от реализации среды выполнения Java.

`static void`	**`setCacheDirectory`**`(`**`File`** `cacheDirectory)`
	
	Задает каталог, в котором должны создаваться файлы кэша.
	
	Параметры:
	
	`cacheDirectory - указание каталога как объекта класса File.`
	
	Выбрасывает:
	`SecurityException - если диспетчер безопасности запрещает доступ к каталогу.`
	
	`IllegalArgumentException - if cacheDirectory не является каталогом.`

`static void`	**`setUseCache`**`(boolean useCache)` Устанавливает флаг, указывающий, следует ли использовать файл кэша на основе диска при создании `ImageInputStream` файлов и `ImageOutputStream`s. При чтении из стандартного `InputStream` может потребоваться сохранить ранее прочитанную информацию в кэше, поскольку базовый поток не позволяет повторно считывать данные. Аналогично,

	при записи в стандартный `OutputStream` может использоваться кэш, позволяющий изменять ранее записанное значение перед его сбросом в конечный пункт назначения. Кэш может находиться в основной памяти или на диске. Установка этого флага в `false` запрещает использование диска для будущих потоков, что может быть выгодно при работе с небольшими изображениями, поскольку устраняются накладные расходы на создание и уничтожение файлов. При запуске это значение устанавливается равным `true`. **Параметры:** `useCache – boolean, указание на то, следует ли использовать файл кэша, в тех случаях, когда это необязательно.`

`static boolean`	**write**(**RenderedImage** im, **String** formatName, **File** output)
	
	Записывает изображение, используя произвольный `ImageWriter`, который поддерживает данный формат, в `File`.
	
	Параметры:
	`im – RenderedImage, который должен быть записан.`
	
	`formatName - String, содержащая неофициальное название формата.`
	
	`output - File, который должен быть записан.`
	
	ВОЗВРАТ:
	`false если соответствующий писатель не найден.`
	
	Выдает:
	`IllegalArgumentException - если какой-либо параметр равен null.`
	
	`IOException - если во время записи возникает ошибка.`

`static boolean`	**write**(**RenderedImage** im, **String** formatName, **ImageOutputStream** output) Записывает изображение, используя произвольный `ImageWriter`, который поддерживает данный формат, в `ImageOutputStream`. Этот метод *не* закрывает предоставленный `ImageOutputStream`после

	завершения операции записи; при желании вызывающий объект обязан закрыть поток. **Параметры:** im – RenderedImage, котороый должен быть записан. formatName - String, содержащая неофициальное название формата. output - объект ImageOutputStream, в который нужно записать. **ВОЗВРАТ:** false если соответствующий писатель не найден. **Выбрасывает:** IllegalArgumentException - если какой-либо параметр равен null. IOException - если во время записи возникает ошибка.
static boolean	**write**(**RenderedImage** im, **String** formatName, **OutputStream** output) Записывает изображение с использованием произвольного ImageWriter, поддерживающего данный формат, в OutputStream. Этот метод *не* закрывает предоставленный OutputStreamпосле завершения операции записи; при желании вызывающий объект обязан закрыть поток. Текущие настройки кэша из getUseCache и getCacheDirectory будут использоваться для управления кэшированием. **Параметры:** im - RenderedImage, который должен быть записан. formatName - String, содержащая неофициальное название формата. output - OutputStreamобъект, в который нужно записать. **ВОЗВРАТ:** false если соответствующий писатель не найден. **Выдает:** IllegalArgumentException - если какой-либо параметр равен null.

IOException - если во время записи возникает ошибка.

Примеры чтения и записи изображения

Пример 1. Чтение изображения с сайта РГРТУ

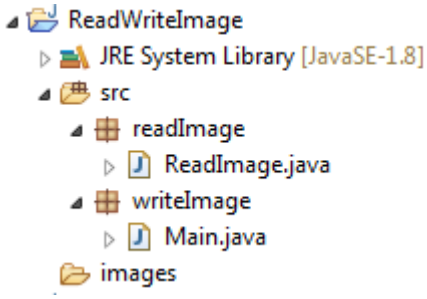
```
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.net.URL;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class ReadImage
{
public static void main( String[] args )
{
BufferedImage image = null;
try {
//Если адрес изображения с заглавной страницы сайта РГРТУ изменился, то укажите новый
//URL
//Интернет должен быть подключен!!!
URL url = new
URL("http://www.rsreu.ru/modules/mod_imgslide/images/fasade_rsreu_2020.jpg");
image = ImageIO.read(url);
} catch (IOException e) {
e.printStackTrace();
}
//Определяем ширину и высоту изображения
int w = image.getWidth();
int h = image.getHeight();

//Создаем фрейм и определяем его размеры
JFrame frame = new JFrame("Фотография с сайта РГРТУ");
frame.setSize(w, h);
//Размещаем изображение на JLabel и выводим его
//Класс IconImage изучите самостоятельно!!!
JLabel label = new JLabel(new ImageIcon(image));
frame.add(label);
frame.setVisible(true);
}
      }
```

Пример 2. Создание и сохранение изображения

Структура проекта




```
package writeImage;

import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
public class Main{
    public static void main(String args[]){
        try{
             //Создаеи изображение
         BufferedImage img = new BufferedImage(500, 500, BufferedImage.TYPE_INT_RGB);
            //Описываем файл
            File f = new File("images/MyFile.png");
            //Это нужно для создания изображения
            int r = 5;
            int g = 25;
            int b = 255;
            int col = (r << 16) | (g << 8) | b;
            //Создаем изображение
            for(int x = 0; x < 500; x++){
                for(int y = 20; y < 300; y++){
                    img.setRGB(x, y, col);
                }
            }
            //Сохраняем изображение в файл
            ImageIO.write(img, "PNG", f);
            //Выводим в консоль полный путь к файлу
            System.out.println(f.getAbsolutePath());
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
//Если изображения не будет видно в каталоге, то в Eclipse
//щелкните вравой кнопкой мыши на каталоге images и выберете пункт Refresh (обновить)
//или нажмите F5
```

ЗАДАНИЕ

Допишите программу из примера 1, так, чтобы полученное с сайта РГРТУ изображение сохранялось в формате png.

Пример 3. Работа с какими типами файлов поддерживается?

Пример демонстрирует, как получить список типов файлов, с которыми может работать ваша система.

```
import javax.imageio.*;

//Чтобы узнать, какой набор установлен для чтения и записи,
//можно спросить об этом класс ImageIO через его методы
//getReaderFormatNames() и getWriterFormatNames()
//или getReaderMIMETypes() и getWriterMIMETypes(),
//если хотите работать непосредственно с MIME типами

public class GetList {
  public static void main(String args[]) {
      //Читатели по имени формата
    String readerNames[] = ImageIO.getReaderFormatNames();
    printlist(readerNames, "Reader names:");
    //Читатели по MIME
    String readerMimes[] = ImageIO.getReaderMIMETypes();
    printlist(readerMimes, "Reader MIME types:");
    //Писатели по имени формата
    String writerNames[] = ImageIO.getWriterFormatNames();
    printlist(writerNames, "Writer names:");
    //Писатели по MIME
    String writerMimes[] = ImageIO.getWriterMIMETypes();
    printlist(writerMimes, "Writer MIME types:");
  }

  private static void printlist(String names[], String title) {
    System.out.println(title);
    for (int i = 0, n = names.length; i < n; i++) {
      System.out.println("\t" + names[i]);
    }
  }
      }
```

Обработка изображений с помощью фильтров

Все классы, предназначенные для обработки изображений, реализуют интерфейс BufferedImageOp. Для преобразования одного изображение в другое после создания объекта-операции необходимо вызвать метод filter:

```
//Создаем объект-операцию над изображением
BufferedImageOp op = …;
//Создаем объект BufferedImage для хранения
//отфильтрованного изображения
BufferedImage filteredImage = new BufferedImage(
image.getWidth(), image.getHeight(), image.getType());
```

```
//Выполняем операцию над исходным изображением image.
//Результат выполнения операции помещаем в filteredImage
op.filter(image, filteredImage);
```

Интерфейс BufferedImageOp реализуют следующие классы:

- AffineTransformOp – выполняет аффинное преобразование пикселей;
- RescaleOp – изменение яркости изображения;
- LookupOp – произвольное отображение выборочных значений;
- ColorConvertOp – преобразование цветового пространства;
- ConvolveOp – свертка (convolution) изображения.

Класс AffineTransformOp

Этот класс выполняет *аффинное преобразование* из 2D-координат в исходном изображении или Raster в 2D-координаты целевого изображения или Raster. Используемый тип интерполяции задается с помощью конструктора либо объектом RenderingHints (hint переводится как подсказка, намек), либо одним из целочисленных типов интерполяции, определенных в этом классе. Должны быть соблюдены следующие ограничения:

- источник и назначение должны быть разными;
- для объектов Raster количество полос в источнике должно быть равно количеству полос в пункте назначения.

Поля класса AffineTransformOp

Модификатор и тип	Поле и описание
`static int`	**`TYPE_BICUBIC`** Бикубический тип интерполяции.
`static int`	**`TYPE_BILINEAR`** Тип билинейной интерполяции.
`static int`	**`TYPE_NEAREST_NEIGHBOR`** Тип интерполяции ближайшего соседа.

Конструкторы класса AffineTransformOp

Конструктор и описание
AffineTransformOp(**AffineTransform** xform, int interpolationType) Создает AffineTransformOp заданное аффинное преобразование и тип интерполяции.
AffineTransformOp(**AffineTransform** xform, **RenderingHints** hints) Создает AffineTransformOp заданное аффинное преобразование.

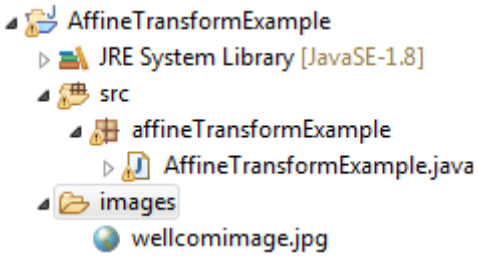
Методы класса AffineTransformOp

Модификатор и тип	Способ и описание
BufferedImage	**createCompatibleDestImage**(**BufferedImage** src, **ColorModel** destCM) Создает обнуленное целевое изображение с правильным размером и количеством полос.
WritableRaster	**createCompatibleDestRaster**(**Raster** src) Создает обнуленное назначение Raster с правильным размером и количеством полос.
BufferedImage	**filter**(**BufferedImage** src, **BufferedImage** dst) Преобразует источник BufferedImage и сохраняет результаты в пункте назначения BufferedImage.
WritableRaster	**filter**(**Raster** src, **WritableRaster** dst) Преобразует источник Raster и сохраняет результаты в пункте назначения Raster.
Rectangle2D	**getBounds2D**(**BufferedImage** src) Возвращает ограничивающую рамку преобразованного изображения.
Rectangle2D	**getBounds2D**(**Raster** src) Возвращает ограничивающую рамку преобразованного назначения.
int	**getInterpolationType**() Возвращает тип интерполяции, используемый этой операцией.
Point2D	**getPoint2D**(**Point2D** srcPt, **Point2D** dstPt) Возвращает местоположение соответствующей точки назначения, заданной точкой в источнике.

RenderingHints	getRenderingHints() Возвращает подсказки рендеринга, используемые этой операцией преобразования.
AffineTransform	getTransform() Возвращает аффинное преобразование, используемое этой операцией преобразования.

Пример использования AffineTransformOp

Структура проекта




```
package affineTransformExample;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.image.*;
import java.awt.geom.AffineTransform;

public class AffineTransformExample extends JFrame {
  ShowLabel label;
  JComboBox shearX, shearY;

String[] shear={"0.0","0.1", "0.2", "0.3", "0.4", "0.5", "0.6", "0.7", "0.8", "0.9"};

  public AffineTransformExample() {
    super("Аффинное преобразование сдвига");
    //Container container = getContentPane();
    label = new ShowLabel();
    this.add(label);

    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(2, 4, 5, 5));

    shearX = new JComboBox(shear);
    shearX.setSelectedItem("0.00");
    panel.add(new JLabel("Сдвиг по X:"));
    panel.add(shearX);
    shearX.addActionListener(new ComboBoxListener());

    shearY = new JComboBox(shear);
    shearY.setSelectedItem("0.0");
      panel.add(new JLabel("Сдвиг по Y:"));
    panel.add(shearY);
    shearY.addActionListener(new ComboBoxListener());
    this.add(BorderLayout.NORTH, panel);
```

```
    setSize(350,300);
    setVisible(true);
  }
    public static void main(String arg[]) {
    new AffineTransformExample();
  }
    //Обработка событий JComboBox
    class ComboBoxListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      JComboBox box = (JComboBox) e.getSource();
        if (box == shearX) {
        label.shearx = Double.parseDouble((String) box.getSelectedItem());
        label.value(true);
        label.filter();
        label.repaint();
      } else if (box == shearY) {
        label.sheary = Double.parseDouble((String) box.getSelectedItem());
        label.value(true);
        label.filter();
        label.repaint();
      }
    }
  }
}

class ShowLabel extends JLabel {
      private static final long serialVersionUID = 1L;
  Image image;
  BufferedImage bufferedImage1, bufferedImage2;
  BufferedImage bufferedImage;
  Graphics2D g2d;
  AffineTransform affineTransform;
  double shearx = 0.0, sheary = 0.0;

  ShowLabel() {
    image = Toolkit.getDefaultToolkit().getImage("images/wellcomimage.jpg");
    MediaTracker mediaTracker = new MediaTracker(this);
    mediaTracker.addImage(image, 1);
    try {
      mediaTracker.waitForAll();
    } catch (Exception e) {}
    createImages();
    affineTransform = new AffineTransform();
  }
    public void createImages() {
      bufferedImage1 = new BufferedImage(image.getWidth(this), image.getHeight(this),
                                                          BufferedImage.TYPE_INT_RGB);
    g2d = bufferedImage1.createGraphics();
    g2d.drawImage(image, 0, 0, this);
    bufferedImage = bufferedImage1;
      bufferedImage2 = new BufferedImage(image.getWidth(this), image.getHeight(this),
                                                          BufferedImage.TYPE_INT_RGB);
  }
    public void value(boolean shear) {
    if (shear) {
      affineTransform.setToShear(shearx, sheary);
      shear = true;
    }
  }
    public void filter() {
  AffineTransformOp affineTransformOp = new AffineTransformOp(affineTransform, null);
```

```
  Graphics2D G2D = bufferedImage2.createGraphics();
   G2D.clearRect(0, 0, bufferedImage2.getWidth(this),bufferedImage2.getHeight(this));
   affineTransformOp.filter(bufferedImage1, bufferedImage2);
   bufferedImage = bufferedImage2;
  }
    public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2D = (Graphics2D) g;
    g2D.drawImage(bufferedImage, 0, 0, this);
  }
}
```

Класс RescaleOp

Класс RescaleOp *масштабирует не изображение, а цвет его пикселей*.

Цвет изменяется в соответствии с выражением $x_{new} = ax + b$. Изменение масштаба цвета при a > 1 приводит к увеличению явкости изображения.

Масштабированные значения выборки обрезаются до минимального или максимального значения, которое можно представить в целевом изображении.

Для растров масштабирование выполняется на диапазонах. Количество наборов констант масштабирования может быть одним, и в этом случае одни и те же константы применяются ко всем полосам, или оно должно равняться количеству полос исходного растра.

Для буферизованных изображений масштабирование работает с цветовыми и альфа-компонентами. Количество наборов масштабирующих констант может быть одним, и в этом случае одни и те же константы применяются ко всем цветовым (но не альфа-компонентам) компонентам. В противном случае количество наборов констант масштабирования может равняться количеству компонентов исходного цвета, и в этом случае масштабирование альфа-компонента (если он присутствует) не выполняется. Если ни один из этих случаев не применим, количество наборов констант масштабирования должно равняться количеству компонентов исходного цвета плюс альфа-компоненты, и в этом случае масштабируются все цветовые и альфа-компоненты.

Источники буферизованных изображений с предварительно умноженными альфа-каналами обрабатываются таким же образом, как и изображения без предварительного умножения. То есть масштабирование выполняется для каждого диапазона для необработанных данных источника BufferedImage без учета того, были ли данные предварительно умножены. Если требуется преобразование цвета в цветовую модель назначения, для этого шага будет учтено предварительно умноженное состояние как источника, так и назначения.

Изображения с IndexColorModel нельзя масштабировать с помощью RescaleOp.

Если в конструкторе указан объект RenderingHints, подсказка цветопередачи и подсказка сглаживания будут использоваться, когда требуется преобразование цвета.

Операция на месте разрешена (т. е. источником и назначением может быть один и тот же объект).

Конструкторы класса RescaleOp

RescaleOp(float[] scaleFactors, float[] offsets, **RenderingHints** hints) Создает новое масштабирование с требуемыми масштабными коэффициентами и смещениями.
RescaleOp(float scaleFactor, float offset, **RenderingHints** hints) Создает новое масштабирование с требуемым масштабным коэффициентом и смещением.

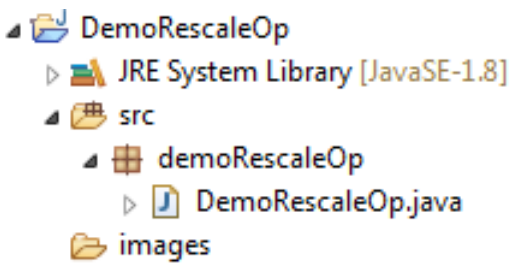
Методы класса RescaleOp

Модификатор и тип	Метод и описание
BufferedImage	**createCompatibleDestImage**(**BufferedImage** src, **ColorModel** destCM) Создает обнуленное целевое изображение с правильным размером и количеством полос.
WritableRaster	**createCompatibleDestRaster**(**Raster** src)

	Создает обнуленное назначение `Raster`с правильным размером и количеством полос, учитывая этот источник.
BufferedImage	**filter**(**BufferedImage** src, **BufferedImage** dst) Изменяет масштаб исходного буферного изображения.
WritableRaster	**filter**(**Raster** src, **WritableRaster** dst) Масштабирует пиксельные данные в исходном растре.
Rectangle2D	**getBounds2D**(**BufferedImage** src) Возвращает ограничивающую рамку масштабированного целевого изображения.
Rectangle2D	**getBounds2D**(**Raster** src) Возвращает ограничивающую рамку масштабированного целевого растра.
`int`	**getNumFactors**() Возвращает количество коэффициентов масштабирования и смещений, используемых при этом изменении масштаба.
`float[]`	**getOffsets**(float[] offsets) Возвращает смещения в заданном массиве.
Point2D	**getPoint2D**(**Point2D** srcPt, **Point2D** dstPt) Возвращает местоположение точки назначения, заданной точкой в источнике.
RenderingHints	**getRenderingHints**() Возвращает подсказки рендеринга для этой операции.
`float[]`	**getScaleFactors**(float[] scaleFactors) Возвращает масштабные коэффициенты в заданном массиве.

Приме использования

Структура проекта



Каталог images вначале пуст. После выполнения программы в нем появятся два файла. Чтобы их увидеть, нажмите в Eclipse F5.

Класс DemoRescaleOp

```
package demoRescaleOp;

import java.awt.image.*;
import java.net.*;
import java.awt.*;
import java.io.*;
import javax.imageio.*;

public class DemoRescaleOp {

    public static void main(String[] args) throws Exception
    {
        // Определяем URL изображения. Попробуйте другое
      URL url= new URL("http://www.rsreu.ru/images/stories/ksapr/sapr2.jpg");

        // Читаем изображение
        BufferedImage image = ImageIO.read(url);
        // Сохраняем исходное изображение на диск
        ImageIO.write(image, "jpg", new File("images/image.jpg"));

        // Устанавливаем значение параметров масштабирования (factors)
        // и смещения (offsets) цвета

      // масштаб для каждой составляющей цвета RGB. Поменяте!
        float[] factors = new float[] {1.45f, 1.45f, 1.45f};
        // смещение для каждой составляющей цвета RGB. Поменяте!
        float[] offsets = new float[] {100.0f, 15.0f, 0.0f};

        // Создаем объект RescaleOp
        RescaleOp rop = new RescaleOp(factors, offsets, null);

        // Фильтруем изображение
        BufferedImage bi = rop.filter((BufferedImage)image, null);

        // Получаем графический контекст изображения
        Graphics2D gbi = (Graphics2D) bi.getGraphics();

        // Добавляем к изображению строку
        gbi.setColor(new Color(255, 255, 255));
        gbi.drawString("Это судьба!!!", 15, 60);

        // Сохраняем отфильтрованное изображение в файл.
        ImageIO.write(bi, "jpg", new File("images/processed.jpg"));
    }
}
      // Если каталог images пуст, то нажмите F5
```

Класс LookupOp

Позволяет создать произвольное изменение выборочных значений. Для этого используется таблица, в которой указаны правила изменения для каждого выборочного значения.

В качестве параметров конструктор LookupOp использует таблицу преобразования LookupTable и необязательную карту с параметрами рисования.

Класс LookupTable является абстрактным и имеет два не абстракных подкласса: ByteLookupTable и ShortLookupTable.

Класс LookupOp реализует операцию поиска от источника к месту назначения. Объект LookupTable может содержать один массив или несколько массивов с учетом приведенных ниже ограничений.

Для растров поиск работает с диапазонами. Количество поисковых массивов может быть одним, и в этом случае один и тот же массив применяется ко всем диапазонам, или же он должен равняться количеству исходных растровых диапазонов.

Для буферных изображений поиск работает с цветовыми и альфа-компонентами. Количество поисковых массивов может быть одним, и в этом случае один и тот же массив применяется ко всем цветным (но не альфа) компонентам. В противном случае количество поисковых массивов может равняться количеству компонентов исходного цвета, и в этом случае поиск альфа-компонента (если он присутствует) не выполняется. Если ни один из этих случаев не применяется, количество массивов поиска должно равняться количеству компонентов исходного цвета плюс альфа-компонентов, и в этом случае поиск выполняется для всех цветовых и альфа-компонентов. Это допускает неравномерное масштабирование многополосных буферных изображений.

Источники буферизованных изображений с предварительно умноженными альфа-данными обрабатываются таким же образом, как и изображения без предварительного умножения для целей поиска. То есть

поиск выполняется для каждого диапазона необработанных данных источника BufferedImage без учета того, были ли данные предварительно умножены. Если требуется преобразование цвета в цветовую модель назначения, для этого шага будет учтено предварительно умноженное состояние как источника, так и назначения.

Изображения с IndexColorModel не могут быть использованы.

Если в конструкторе указан объект RenderingHints, подсказка цветопередачи и подсказка сглаживания могут использоваться, когда требуется преобразование цвета.

Этот класс позволяет источнику быть таким же, как и получателю.

Конструктор LookupOp

Конструктор и описание
LookupOp(**LookupTable** lookup, **RenderingHints** hints) Создает LookupOp объект с учетом таблицы поиска и объекта RenderingHints, который может быть null. **Параметры:** lookup - указанный LookupTable hints – указанный RenderingHints, или null

Методы LookupOp

Модификатор и тип	Метод и описание
BufferedImage	**createCompatibleDestImage**(**BufferedImage** src, **ColorModel** destCM) Создает обнуленное целевое изображение с правильным размером и количеством полос. Если destCM - null, ColorModel, то будет использоваться подходящая цветовая модель. **Параметры:** src - исходное изображение для операции фильтрации destCM - ColorModel места назначения, которая может быть null. **ВОЗВРАТ:**

отфильтрованное BufferedImage.

WritableRaster **createCompatibleDestRaster**(**Raster** src)

Создает обнуленное назначение Rasterс правильным размером и количеством полос, учитывая этот источник.

Параметры:
src – Raster, подлежащий преобразованию

ВОЗВРАТ:
обнуленный Raster - место назначения.

BufferedImage **filter**(**BufferedImage** src, **BufferedImage** dst)

Выполняет операцию поиска в BufferedImage.

Если цветовая модель в исходном изображении отличается от цветовой модели в целевом изображении, пиксели будут преобразованы в целевом изображении. Если целевым изображением является null, BufferedImage будет создан соответствующий ColorModel. IllegalArgumentException Может быть выдан, если количество массивов в LookupTable не соответствует ограничениям, указанным в комментарии к классу выше, или если исходное изображение имеет IndexColorModel.

Параметры:
src – BufferedImage, подлежащий фильтрации

dst – BufferedImage место, в котором хранятся результаты операции фильтрации

ВОЗВРАТ:
отфильтрованный BufferedImage.

Выбрасывает:
IllegalArgumentException - если количество массивов в LookupTablene соответствует ограничениям, описанным в комментариях к классу, или если исходное изображение имеет IndexColorModel.

WritableRaster **filter**(**Raster** src, **WritableRaster** dst)

Выполняет операцию поиска в Raster.

Если адресат Raster null Raster указан, то будет создан новый. IllegalArgumentException может быть выброшено, если источник Raster и место назначения Raster не имеют одинакового количества диапазонов или если количество массивов в LookupTable не соответствует ограничениям, указанным в комментарии к классу выше.

Параметры:
src – Raster - источник для фильтрации

dst - WritableRaster - место назначения для отфильтрованного src

ВОЗВРАТ:

отфильтрованный WritableRaster.

Выбрасывает:

IllegalArgumentException - если растры источника и назначения не имеют одинакового количества диапазонов или количество массивов в LookupTable не соответствует ограничениям, описанным в комментариях к классу.

Rectangle2D	**getBounds2D**(**BufferedImage** src) Возвращает ограничивающую рамку отфильтрованного целевого изображения. Поскольку это не геометрическая операция, ограничивающая рамка не изменяется. **Параметры:** src - BufferedImage, подлежащий фильтрации **ВОЗВРАТ:** границы изображения с отфильтрованной четкостью.
Rectangle2D	**getBounds2D**(**Raster** src) Возвращает ограничивающую рамку отфильтрованного целевого растра. Поскольку это не геометрическая операция, ограничивающая рамка не изменяется. **Параметры:** src - Rasterподлежащий фильтрации **ВОЗВРАТ:** границы отфильтрованного Raster.
Point2D	**getPoint2D**(**Point2D** srcPt, **Point2D** dstPt) Возвращает местоположение точки назначения с учетом точки в источнике. Если dstPt не nullто, он будет использоваться для хранения возвращаемого значения. Поскольку это не геометрическая операция, то srcPt будет равно dstPt. **Параметры:** srcPt - Point2D, представляющий точку на исходном изображении dstPt - aPoint2D, который представляет местоположение в месте назначения **ВОЗВРАТ:** Point2D в месте назначения, который соответствует указанной точке в источнике.
RenderingHints	**getRenderingHints**() Возвращает подсказки по рендерингу для этой операции. **ВОЗВРАТ:**

	Объект RenderingHints, связанный с этой операцией.
LookupTable	getTable() Возвращает LookupTable.

Класс ColorConvertOp

Этот класс выполняет попиксельное преобразование цвета данных в исходном изображении. Результирующие значения цвета масштабируются с точностью до целевого изображения. Преобразование цвета может быть задано с помощью массива объектов ColorSpace или массива объектов ICC_Profile.

Если источником является буферизованное изображение с предварительно умноженным альфа-значением, цветовые компоненты делятся на альфа-компонент перед преобразованием цвета. Если конечным объектом является буферизованное изображение с предварительно умноженным альфа-значением, цветовые компоненты умножаются на альфа-компонент после преобразования. Растры обрабатываются как не имеющие альфа-канала, т. е. Все полосы являются цветными полосами.

Если в конструкторе указан объект RenderingHints, подсказка цветопередачи и подсказка сглаживания могут использоваться для управления преобразованием цвета.

Источником и назначением может быть один и тот же объект.

Кострукторы ColorConvertOp

Конструктор и описание
ColorConvertOp(ColorSpace srcCspace, ColorSpace dstCspace, RenderingHints hints) Создает новый ColorConvertOp из двух объектов цветового пространства. Аргумент RenderingHints может быть нулевым. Эта операция в первую очередь полезна для вызова метода filter для растров, и в этом случае два цветовых пространства определяют операцию, которая должна выполняться для растров. В этом случае количество полос в исходном растре должно соответствовать количеству компонентов в srcCspace, а количество

полос в целевом растре должно соответствовать количеству компонентов в dstCspace. Для изображений с буферизацией два цветовых пространства определяют промежуточные пространства, через которые источник преобразуется перед преобразованием в целевое пространство.

Параметры:

`srcCspace` - источник ColorSpace

`dstCspace` - пункт назначения ColorSpace

`hints` - RenderingHintsобъект, используемый для управления преобразованием цвета, или null

Выдает:

`NullPointerException` - если либо srcCspace, либо dstCspace равно нулю

ColorConvertOp(**ColorSpace** cspace, **RenderingHints** hints)

Создает новый ColorConvertOp из объекта ColorSpace.

Аргумент RenderingHints может быть нулевым. Эта операция может использоваться только с BufferedImages и в первую очередь полезна, когда вызывается метод `filter` с целевым аргументом null. В этом случае цветовое пространство определяет цветовое пространство назначения для назначения, созданного методом фильтра. В противном случае цветовое пространство определяет промежуточное пространство, в которое преобразуется источник перед преобразованием в целевое пространство.

Параметры:

`cspace` - определяет пункт назначения ColorSpaceили промежуточный ColorSpace

`hints` - RenderingHintsобъект, используемый для управления преобразованием цвета, или null

Выбрасывает:

`NullPointerException` - если cspace равно нулю

ColorConvertOp(**ICC_Profile**[] profiles, **RenderingHints** hints)

Создает новый ColorConvertOp из массива ICC_Profiles.

Аргумент RenderingHints может быть нулевым. Последовательность профилей может включать в себя профили, которые представляют цветовые пространства, профили, которые представляют эффекты и т.д. Если вся последовательность не представляет четко определенного преобразования цвета, генерируется исключение.

Для буферных изображений, если цветовое пространство исходного буферного изображения не соответствует требованиям первого профиля в массиве, первое преобразование выполняется в соответствующее цветовое пространство. Если цветовое пространство буферного изображения назначения не соответствует требованиям последнего профиля в массиве, последнее преобразование выполняется в цветовое пространство назначения.

Для растров количество полос в исходном растре должно соответствовать требованиям первого профиля в массиве, а количество полос в целевом растре должно соответствовать требованиям последнего профиля в массиве. Массив должен содержать как минимум два элемента, иначе вызов метода filter для растров вызовет исключение IllegalArgumentException.

Параметры:

`profiles` - массив `ICC_Profile`объектов

`hints` – объект `RenderingHints`, используемый для управления преобразованием цвета, или `null`

Выбрасывает:

`IllegalArgumentException` - когда последовательность профилей не определяет четко определенное преобразование цвета

`NullPointerException` - если значение `profiles` равно `null`

`ColorConvertOp`(**`RenderingHints`** `hints`)

Создает новый ColorConvertOp, который преобразует исходное цветовое пространство в целевое цветовое пространство.

Аргумент RenderingHints может быть нулевым. Эта операция может использоваться только с BufferedImages и будет преобразовывать из цветового пространства исходного изображения в цветовое пространство назначения. Целевой аргумент метода filter не может быть указан как null.

Параметры:

`hints` – объект `RenderingHints`, используемый для управления преобразованием цвета, или `null`

Методы ColorConvertOp

Модификатор и тип	Способ и описание
`BufferedImage`	**`createCompatibleDestImage`**(**`BufferedImage`** `src`, **`ColorModel`** `destCM`) Создает обнуленное целевое изображение с правильным размером и количеством полос, учитывая этот источник.
`WritableRaster`	**`createCompatibleDestRaster`**(**`Raster`** `src`) Создает обнуленный целевой растр с правильным размером и количеством полос, учитывая этот источник.
`BufferedImage`	**`filter`**(**`BufferedImage`** `src`, **`BufferedImage`** `dest`) Преобразует в цвет исходное буферное изображение.
`WritableRaster`	**`filter`**(**`Raster`** `src`, **`WritableRaster`** `dest`) Преобразование цвета данных изображения в исходном растре.
`Rectangle2D`	**`getBounds2D`**(**`BufferedImage`** `src`) Возвращает ограничительную рамку назначения с учетом этого источника.
`Rectangle2D`	**`getBounds2D`**(**`Raster`** `src`)

	Возвращает ограничительную рамку назначения с учетом этого источника.
ICC_Profile[]	getICC_Profiles() Возвращает массив ICC_Profiles, используемый для построения этого ColorConvertOp.
Point2D	getPoint2D(Point2D srcPt, Point2D dstPt) Возвращает местоположение точки назначения, заданной точкой в источнике.
RenderingHints	getRenderingHints() Возвращает подсказки рендеринга, используемые этой операцией.

Класс ConvolveOp

Класс реализует операцию свертки (convolution), которая определяет значение цвета точки в зависимости от цветов окружающих точек. Пусть точка с координитами (x, y) имеет цвет C(x, y). Составляем массив весовых коэффициентов, например, из 9 чисел: $\begin{pmatrix} w0 & w1 & w2 \\ w3 & w4 & w5 \\ w6 & w7 & w8 \end{pmatrix}$. Тогда новое значение цвета в точке (x, y) вычисляется следующим образом:

$$C_{new}(x,y) =$$

$$\begin{aligned} &w0 * C(x-1,y-1) + \quad w1 * C(x,y-1) + \quad w2 * C(x+1,y-1) + \\ &w3 * C(x-1,y) + \quad w4 * C(x,y) + \quad w5 * C(x+1,y) + \\ &w6 * C(x-1,y+1) + \quad w7 * C(x,y+1) + \quad w8 * C(x+1,y+1) \end{aligned}$$

Определяя различные значения весовых коэффициентов можно получать различные визуальные эффекты, усиливая или уменьшая влияние соседних точек.

Если сумма весовых коэффициентов равна 1, то интенсивность цвета на результирующем изображении останется прежней.

Можно использовать различные массивы весовых коэффициентов, нужно только чтобы размерности массива коэффициентов были нечетными.

Свертка выполняется следующим образом:

1) Определяем массив весов, например: float[] w = {0, -1, 0, -1, 5, -1, 0, -1, 0};
2) Создаем экземпляр класса Kernel – ядра свертки: Kernel kern = new Kernel(3, 3, w)
3) Создаем объект класса ConvolveOp с этим ядром: ConvolveOp conv = ConvolveOp(kern);
4) Применяем метод filter() класса ConvolveOp: conv.filter(bi, newbi), где bi и newbi – исходное и отфильтрованное изображение класса BufferedImage.

Класс ConvolveOp работает с данными BufferedImage, в которых цветовые компоненты предварительно умножаются на альфа-компонент. Если исходное BufferedImage имеет альфа-компонент, а цветовые компоненты не умножаются предварительно на альфа-компонент, то данные предварительно умножаются на альфа-компонент перед сверткой. Если в месте назначения есть цветовые компоненты, которые не умножаются предварительно, то альфа-значение разделяется перед сохранением в пункте назначения (если альфа равна 0, цветовые компоненты устанавливаются равными 0). Если у назначения нет альфа-компонента, то результирующая альфа отбрасывается после первого разделения ее на цветовые компоненты.

Растры обрабатываются как не имеющие альфа-канала. Если вышеуказанная обработка альфа-канала в BufferedImages нежелательна, ее можно избежать, получив растр исходного BufferedImage и используя метод filter этого класса, который работает с растрами.

Если в конструкторе указан объект RenderingHints, подсказка цветопередачи и подсказка сглаживания могут использоваться, когда требуется преобразование цвета.

Источником и местом назначения не может быть один и тот же объект.

Поля класса ConvolveOp

Модификатор и тип	Поле и описание
static int	**EDGE_NO_OP** Пиксели на краю исходного изображения копируются в соответствующие пиксели в месте назначения без изменений.
static int	**EDGE_ZERO_FILL** Пиксели на краю целевого изображения обнуляются. Это значение используется по умолчанию.

Конструкторы класса ConvolveOp

Конструктор и описание
ConvolveOp(**Kernel** kernel) Создает свертку с заданным ядром.
ConvolveOp(**Kernel** kernel, int edgeCondition, **RenderingHints** hints) Создает ConvolveOp с учетом ядра, граничного условия и объекта RenderingHints (который может быть нулевым).

Методы класса ConvolveOp

Модификатор и тип	Метод и описание
BufferedImage	**createCompatibleDestImage**(**BufferedImage** src, **ColorModel** destCM) Создает обнуленное целевое изображение с правильным размером и количеством полос.
WritableRaster	**createCompatibleDestRaster**(**Raster** src) Создает обнуленный целевой растр с правильным размером и количеством полос, учитывая этот источник.
BufferedImage	**filter**(**BufferedImage** src, **BufferedImage** dst) Выполняет свертку для буферизованных изображений. Каждый компонент исходного изображения будет свернут (включая альфа-компонент, если он присутствует). Если цветовая модель в исходном изображении отличается от цветовой модели в целевом изображении, пиксели будут преобразованы в цветовую модель в целевом изображении. Если целевому изображению присвоено

значение null, будет создано BufferedImage с использованием исходной цветовой модели. Исключение IllegalArgumentException может быть вызвано, если источник совпадает с местом назначения.

Параметры:

src - источник BufferedImage для фильтрации

dst - место назначения BufferedImage для отфильтрованного src

ВОЗВРАТ:

отфильтрованный BufferedImage

Выбрасывает:

NullPointerException - если srcесть null

IllegalArgumentException - если src равно dst

ImagingOpException - если srcн евозможно отфильтровать

WritableRaster	**filter**(**Raster** src, **WritableRaster** dst)

Выполняет свертку для растров.

Каждая полоса исходного растра будет свернута. Источник и место назначения должны иметь одинаковое количество диапазонов. Если целевой растр имеет значение null, будет создан новый растр. Исключение IllegalArgumentException может быть выброшено, если источник совпадает с местом назначения.

Параметры:

src - источник Raster для фильтрации

dst - место назначения WritableRaster для отфильтрованного src

ВОЗВРАТ:

отфильтрованный WritableRaster

Иыбрасывает:

NullPointerException - если src - null

ImagingOpException - если src и dst не имеют одинакового количества полос

ImagingOpException - если src невозможно отфильтровать

IllegalArgumentException - если src равно dst

Rectangle2D	**getBounds2D**(**BufferedImage** src)

Возвращает ограничивающую рамку отфильтрованного целевого изображения.

Rectangle2D	**getBounds2D**(**Raster** src)

Возвращает ограничивающую рамку отфильтрованного целевого растра.

int	**getEdgeCondition**()

	Возвращает граничное условие.
Kernel	**getKernel**() Возвращает ядро.
Point2D	**getPoint2D**(**Point2D** srcPt, **Point2D** dstPt) Возвращает местоположение точки назначения с учетом точки в источнике.
RenderingHints	**getRenderingHints**() Возвращает подсказки рендеринга для этой операции.

Пример использования. Фильтр размытия

```
import java.awt.*;
import java.awt.image.*;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import javax.imageio.ImageIO;
import java.awt.event.*;

public class ConvolveDemo extends Frame{
	private static final long serialVersionUID = 1L;
	private BufferedImage bi;
// Конструктор
public ConvolveDemo(String s) {
	super (s) ;
	URL url = null;
// Адрес изображения на сайте РГРТУ
	try {
		url = new
URL("http://www.rsreu.ru/modules/mod_imgslide/images/fasade_rsreu_2020.jpg");
	} catch (MalformedURLException e) {
		e.printStackTrace();
	}

//Читаем изображение с сайта РГРТУ
BufferedImage img = null;
try {
	img = ImageIO.read(url);
} catch (IOException e1) {
	e1.printStackTrace();
}
// Создаем объект BufferedImage
bi = new BufferedImage(img.getWidth(null),
img.getHeight(null),BufferedImage.TYPE_INT_RGB);
// Выводим
Graphics2D big = bi.createGraphics();
big.drawImage(img, 0, 0, this);
}

public void paint(Graphics g){
Graphics2D g2 = (Graphics2D)g;
int w = this.getSize().width;
```

```
int bw = bi.getWidth(null);
int bh = bi.getHeight(null);

BufferedImage bimg = new BufferedImage(bw, bh, BufferedImage.TYPE_INT_RGB);
//Матрица фильтра размытия для ядра свёртки
float[] wl = {
0.11111111f, 0.11111111f, 0.11111111f,
0.11111111f, 0.11111111f, 0.11111111f,
0.11111111f, 0.11111111f, 0.11111111f };
// Ядро свертки
Kernel kern = new Kernel(3, 3, wl);
// Обънкт ConvolveOp
ConvolveOp cop = new ConvolveOp(kern, ConvolveOp.EDGE_NO_OP, null);
// Применение свертки
cop.filter(bi, bimg) ;
// Выводим исходное и размытое изображения
g2.drawImage(bi, null, 0, 0);
g2.drawImage(bimg, null, w/2, 0);
}

public static void main(String args[]){
      Frame f = new ConvolveDemo("ConvolveDemo - Исходное и размытое изображения");
      f.addWindowListener(new WindowAdapter(){
                  public void windowClosing(WindowEvent e) {
                  System.exit(0);
                  }
      }
                                          );
      f.setSize(800,300);
      f.setResizable(false);
      f.setVisible(true);
}
}
```

ЗАДАНИЕ. Реализуйте фильтры со следующими ядрами. Результат применения каждого фильтра сохраните в отдельном файле.

1) Фильтр размытия Гаусса с ядром

0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

2) Фильт размытия с ядром

0,05	0,05	0,05
0,05	0,60	0,05
0,05	0,05	0,05

3) Фильтр повышения резкости с ядром

$$\frac{1}{10}\cdot\begin{Vmatrix} -1 & -2 & -1 \\ -2 & 22 & -2 \\ -1 & -2 & -1 \end{Vmatrix}.$$

4) Фильтр выделения границ с ядром

0	1	0
1	-3	1
0	1	0

5) Фильтр тиснения с ядром

-2	-1	0
-1	1	1
0	1	2

6) Акварельный фильтр.

Выполняется в 2 шага.

Шаг 1. Фильтр с ядром

$$\frac{1}{16}\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}$$

Шаг 2. Фильт с ядром

$$\begin{vmatrix} -0,5 & -0,5 & -0,5 \\ -0,5 & 5 & -0,5 \\ -0,5 & -0,5 & -0,5 \end{vmatrix}$$